Interactive Computer Graphics with the UNIX Time-Sharing System

By Thomas E. Ferrin and Robert Langridge

23 October 1978

(Revised: 2 April 1979)

University of California, San Francisco
Computer Graphics Laboratory
Department of Pharmaceutical Chemistry
School of Pharmacy
San Francisco, California 94143

Interactive Computer Graphics with the UNIX Time-Sharing System*

Thomas E. Ferrin and Robert Langridge,
Computer Graphics Laboratory
Department of Pharmaceutical Chemistry
School of Pharmacy
University of California,
San Francisco

The UNIX† time-sharing system is a powerful and efficient multiuser operating system possessing many unique and attractive advantages. In order to fully support an interactive graphics facility, however, several minor extensions were made to provide the necessary high performance 'real-time' environment. These enhancements to the UNIX system are described together with the implementation details for the Picture System 2 Graphic Subroutine Package.

Introduction:

The first application of interactive computer graphics to molecular structure representation appeared well over a decade ago and has been the subject of many excellent reviews.[1-3] Great advances have been made in the capabilities of these early graphic systems and the rapidly decreasing cost of hardware has made such systems available to many scientific investigators. Computer science in general has made large strides and the once elite, large-machine based time-sharing systems have been refined and improved to the point where they now use hardware that almost any laboratory can afford. The advantages of computer time-sharing are so overwhelming when compared to dedicated systems that functions which historically were not compatible with a time-sharing environment (such as interactive computer graphics) are now being integrated into these systems.

The Computer Graphics Laboratory at the University of California, San Francisco‡ uses the UNIX operating system to support a high performance interactive graphics facility for display of three-dimensional molecular models. The primary interests are the structures, functions, and interactions of complex biological molecules, particularly those related to drug interactions. Many of the structures, such as proteins and amino acids, are of sufficient size (ca. 100,000 molecular weight) to make the construction of physical models extremely time consuming and awkward. As the models become even larger and their physical weight grows to the order of several kilograms, stability becomes a crucial problem. Storage of such structures is also difficult and models filling a small room are not uncommon. In addition to these disadvantages, physical models are difficult to manipulate with any degree of precision, particularly if one wishes to make adjustments to an interior region of the structure. Lastly, emphasis on a particular area of interest and exclusion of other portions may be completely impossible with physical models since pieces of the structure would be left disconnected and unsupported.

Computer graphics provides solutions to these and other problems while providing many other advantages. An even greater benefit is realized when the observer can play an active role in how he wishes to view and manipulate the model. Such capabilities are currently provided only through an interactive three-dimensional display system such as the Evans & Sutherland Computer Corporation's Picture System 2.[4]

The Picture System 2 (figure 1) is a calligraphic display system incorporating a matrix arithmetic processor *(MAP)* designed specifically for graphics oriented operations on two- and three-dimensional data. The data may be translated, rotated and scaled before it is mapped onto a user definable viewport area for final display on a 25.5 cm cathode ray tube (P4 phosphor). A specification for the range of allowable data values creates a *window* into the data space so that only the regions of immediate interest are displayed while all other line segments are *clipped* and hence not visible. The window specification also includes information specifying the position of the observer so that a true perspective transformation can be applied to any three-dimensional data. Since the viewport specification includes information for a gradient on

* The ideas detailed in this paper were presented on 21 March 1978 at the annual UNIX User's Group Meeting held at Columbia University, New York.

† UNIX is a trademark of Bell Laboratories.

‡ The *University of California, San Francisco Computer Graphics Laboratory* is funded primarily by the National Institutes of Health, Division of Biotechnology Research Resources and also by the University of California, San Francisco. The director and principal investigator of the Computer Graphics Laboratory is Professor Robert Langridge, Professor of Pharmaceutical Chemistry and of Biochemistry and Biophysics. The computer facility manager is Mr Thomas Ferrin.

displayed line intensity also, depth cueing through intensity variation, as well as picture perspective, provides realistic two-dimensional representations of three-dimensional structures. A hardware character generator is provided for generating any of eight different sizes of alpha-numeric characters and may also be programmed by the user for generating special purpose symbols. Since the display system contains its own refresh buffer and controller, flicker-free displays of several thousand vectors are possible regardless of the rate at which the displayed picture is updated by the host computer.

User interaction with the display is through the manipulation of several interactive devices connected directly to the display system. These include a two-dimensional data tablet, eight continuous rotation control dials, two three-dimensional action joysticks, a bank of 16 toggle switches and incandescent lamps and an alpha-numeric keyboard. The exact function of all the devices is dependent upon the control program executing in the host computer.

The Picture System 2 thus makes it possible to generate complex line drawing pictures and to transform these pictures rapidly. In molecular modeling this entails generating a series of vectors representing the bonds between atoms, optionally labeling the end points of the vectors with atom name or group identification, and spatially orienting the picture to the desired viewing position. This last function is repeated many times each second so that by using the control dials or joysticks the picture may be manipulated to any new position on the viewing screen. By entering commands on the keyboard, selected parts of a molecule may be made visible or singled out for further manipulation (such as local rotations around selected atomic bonds). The current implementation of application software allows up to four independent models to be displayed simultaneously.

The host processor for the graphics system is a Digital Equipment Corporation PDP-11/70 computer, currently equipped with 448 Kbytes of main memory. Several pieces of additional hardware have been acquired specifically to complement the high performance graphics display. These include a 300 Mbyte disk drive (35 msec average access with 1.2 Mbyte/sec transfer rate), 1 Mbyte of bulk core storage configured to appear as a fixed head disk to the central processor (10 μsec access, 1 Mbyte/sec transfer rate), a 16 line direct memory access asynchronous multiplexer for terminal support, a high speed electrostatic printer/plotter and a variety of other peripherals. A 4800 baud synchronous communications dial-up link to a Control Data Corporation 7600 computer at Lawrence Berkeley Laboratory for remote job entry of large scale numerical calculations has also been implemented to provide additional computational support.

The decision to use the UNIX operating system[5,6] for all program development and support of the graphics display system was made early in the design phase of the laboratory. The reasons are multifold: UNIX provides a friendly and efficient environment that is highly productive for program development and documentation; a rich set of programming languages is available for use so that there is no hesitation about choosing another language if a particular application justifies it; the system itself is written almost entirely in a high level language thus making it easy to understand, maintain and modify; and, finally, UNIX is a very popular system not only at Bell Laboratories but among colleges and universities, particularly at departments of computer science, thus assuring its longevity and inspiring an active user community to cultivate exchange of ideas and programs Disadvantages of choosing UNIX were primarily

(i) no support of the Evans & Sutherland (or any other commercially available) high performance graphics display system and,

(ii) the potential problems associated with supporting an interactive graphics process in a time-sharing environment.

Reviewing our decision two years later has produced the overwhelming conviction that UNIX was far and away the best choice.

UNIX represents a rather radical departure from previously available mini-computer operating systems. Of the approximately 10,000 lines of code that make up the kernel, over 9,000 of them are written in the C programming language[7] developed at Bell Laboratories C provides many of the capabilities of high level languages while allowing the programmer contact with the elementary architecture of the computer ('register' is a fundamental data type, for instance). This allows much of the power, flexibility, and efficiency normally associated only with assembly language programming while keeping the program structure, data structure, and diagnostic features found in high level languages. Code generated by the C compiler is efficient and modest in memory requirement — important advantages both to operating systems and high performance graphic subroutines.

Evans & Sutherland provides no support for their graphics hardware under UNIX. Thus, the first major programming project was to develop a set of subroutines for use with the Picture System 2. It was decided that while most of the features available with the subroutine package that Evans & Sutherland did provide (for use with DEC's RT-11 and RSX-11M

operating systems) were desirable, it would be difficult to transport these to UNIX since they were written entirely in assembly language, and were both complex and difficult to understand. In addition, the existing subroutines were based on an earlier model of the Picture System 2 display and did not provide for some of the more powerful features available with the current display hardware. We therefore rewrote all the graphic subroutines in the C language, implementing the additional routines necessary for the advanced hardware features and correcting some of the more dubious design choices in the original routines. This project was undertaken in the spring of 1977 in collaboration with the Chemistry Department Computer Facility of the University of California, San Diego. (That facility is also running under UNIX and was interested in converting their early model Picture System from RT-11 to UNIX during the same time period.)

The "Graphics Subroutine Package" development project has been highly successful.[8] This package is not only a fraction of the size of the original source listings, but is also much more understandable and hence maintainable. In addition, several efficiency enhancements have been made to improve throughput as well as to support the additional hardware features of the Picture System 2.

The entire set of routines (about 70 in all) is organized in a three-level hierarchal fashion and consists of approximately 4700 line of code, only 400 of which are written in assembly language. (These later routines are necessary either because of peculiarities in the PDP-11 instruction set or because the ultimate in execution speed is desirable). The three levels consist of

(i) a well defined user interface which is functionally divided into a basic set of device independent routines,

(ii) a set of routines intended for use internally by the above functions which implement low-level procedures and form the device dependent interface, and

(iii) a system level device driver.

(For an excellent discussion on the design principles of a graphics subroutine package see reference 9.)

The user interface consists of sets of routines to implement graphics primitives, windowing functions, transformation functions, picture segmentation functions, and input functions. Graphics primitives include the procedures used to specify the lines, dots, characters, etc. that actually make up the picture. Several similar procedures provide the user with a flexible and comprehensive means of specifying the most common functions. For instance, *lineto(x,y), line(dx,dy),* and *draw2d(array, num,...)* are all routines for drawing a straight line. However, in the first cal the *x,y* pair are absolute coordinates, in the second they are coordinates relative to the current position, and in the last call they are part of an array of *x,y* coordinate pairs, either absolute or relative (distinguished by an additional parameter). Thus the user may choose which routine best suits the given data and need not be concerned with translating the data into the "correct" representation for a particular subroutine call.

The low level internal procedures include such functions as the buffering of display commands and data in order to decrease the amount of overhead associated with UNIX system calls. The technique involves declaring an array to hold data from several system calls and then passing the array of data with one call. With the potential for a large number of small word count data transfers to the Picture System, this mechanism saves a substantial amount of system overhead. (Even on the PDP-11/70 each UNIX system call takes a minimum of 320 μsec.) This level of functionality is incorporated into the software package in such a way as to be transparent to the user. Once the location and size of the *write buffer* data array is declared the display routines invisibly buffer output to the Picture System, flushing the buffer whenever necessary.

Not all user callable routines are device independent, however. While this may be undesirable from a portability point of view, it allows many of the powerful display hardware capabilities to be fully utilized. The ability to process commands and data through the Picture System 2 MAP at full hardware speed and then *read back* all or selected special portions of the transformed data coordinates for further calculations (such as interatomic distances or bond angles) is an example of such a routine. This powerful feature is used extensively by some of the molecular modeling programs.[10,12-14]

Probably the most important aspect of the new graphic subroutine package is its syntactic compatibility with the Evans & Sutherland version. In particular, much of the same user's manual[11] is used as reference for both the UNIX version and the Evans & Sutherland version. There are minor exceptions to this, most of which are due to either the time-sharing nature of UNIX, the subroutine calling linkage details of the C language, or to take more advantage of hardware capabilities. The compatibility aspect, however, is important both in easing the burden of support documentation and in providing a smooth transition for other Picture System 2 installations moving to the

UNIX operating system.§ User documentation for the new package consists of supplying the standard Evans & Sutherland User's Manual along with about a dozen and a half replacement pages to reflect the new or changed features.

The UNIX/PS2GSP has been the foundation of several major user application programs for use in displaying and manipulating molecular models. These include the Molecular Interactive Display System (MIDS),[12] Protein Interactive Graphics System (PIG),[13] and Picture Builder (Bild).[14] Each of these is a large multi-process program that is used on a production level basis during normal laboratory hours on a system otherwise lightly to moderately loaded with other users performing program development and text editing. There is a noticeable impact on system throughput and performance when an interactive graphics program is running, but the degradation is more than tolerable for the other users and demonstrates both the efficiency of the overall system and the successfulness of a multi-user configuration even when a high performance interactive graphics display is in use.

All of the work described so far was done within the confines of the standard version of the UNIX operating system. For those readers unfamiliar with UNIX, reference 6 provides an in depth description of the UNIX system, ranging from the operating system internals on up through several application programs. Briefly, UNIX is a powerful and compact multiuser operating system available for the DEC PDP-11 series of minicomputers. (UNIX also now operates on the Interdata 8/32 and the DEC VAX-11/780 computers.) The system supports a wide variety of high level programming languages and presents a friendly and highly productive environment for the user. As previously mentioned, the system itself is written in the C programming language and hence is easy to maintain and modify. Thus, the Picture System 2 was interfaced to the system through an ordinary device driver (albeit large in size but written in the C language, of course) in much the same way as any other peripheral device driver. In order to gain increased performance from the entire configuration, however, several extensions were made to both the UNIX software and the PDP-11/70 hardware configuration.

The most important of these extensions has been to the UNIX process scheduler. In order to provide consistent and reliable "real-time" response for the currently executing graphics process, the scheduler was modified to treat this process in a special way.

Real-time is taken here to mean quick enough and frequent enough response to provide smooth user interaction with the graphics display. Since the Picture System 2 hardware handles refresh of the display screen automatically, only the picture *update* rate is of concern. An update rate of 15-20 frames per second is adequate to provide smooth motion of the displayed picture. Thus, the host cpu must schedule the graphics process for execution at a minimum of every 40-70 msec.

UNIX allows some modification of user process priority within a limited range (the *nice(...)* system call) to provide both "background" and "high priority" processes in the normal time-sharing environment. This standard mechanism proved unacceptable for use with interactive graphics, since occasionally the graphics process did not receive control of the cpu often enough, leading to momentary hesitations and "jitter" of the displayed picture. This was very distracting to the user and contributed to a loss of synchronization between the change in values from the interactive devices controlled by the user and subsequent change in the displayed picture. Cause of this decreased response time was typically due to the execution of a higher priority "kernel mode" process.

[In UNIX a kernel process cannot be asynchronously preempted. That is, a process which is executing in kernel mode cannot be arbitrarily suspended by another higher priority process (distinct from a hardware interrupt). This strategy vastly simplifies the data concurrency problems that have frequently plagued other operating systems.]

The enhancements to the system process scheduler are in three distinct areas. The first and most obvious is to insure that the graphics process is never swapped out of main memory. This is a straightforward modification since the mechanism to accomplish this is already utilized by another process in the UNIX system. A bit in the per process data structure *("SSYS")* indicates that a process (normally only the system's swap scheduler) is not eligible for swapping. A new system call *(rtp(...))* was added that sets this bit for the currently executing process, providing it has the necessary privileges, of course. This in itself does not *guarantee* that the graphics process is always core resident, since the dynamic stack allocation mechanism in UNIX may require that a process be swapped out in order to satisfy additional memory requirements, but it does guarantee that the graphics process is not swapped out in order that another arbitrary process can execute, regardless of its priority.

§ There are currently six other installations utilizing the software subroutines developed at our laboratory in a variety of applications.

The second modification, implemented with the same *rtp(...)* system call, sets a flag indicating to the UNIX process scheduler that the currently executing process is to be considered a "real-time" process and is to receive top priority when scheduling the cpu. Additional code was added in the process scheduler that tests this flag (actually a pointer to the entry in the process table of the "real-time process") and executes the privileged process whenever it is in a runnable condition. Since the display system hardware and Picture System 2 device driver provide the necessary timing for the selected 15-20 frames per second update rate, the graphics process is synchronized with the updates of the display picture and executes only at the selected rate. As long as the graphics process does not utilize 100% of the available cpu cycles (not even feasible, since I/O within UNIX is synchronous in nature), excess cpu time is available for other users. This technique has been very successful and provides nearly all the real-time scheduling requirements needed to support interactive graphics in our time-sharing environment.

Occasional problems did still arise with the original modifications detailed above, however. At times, random pauses and hesitations of the otherwise smoothly changing picture were sometimes noticed, particularly during periods of heavy operating system activity. This indicated that occasionally the graphics process was still not receiving control of the cpu often enough to meet the picture update requirements. After much time consuming detective work, the cause of this problem was located and corrected.

It was mentioned above that the UNIX kernel is not preemptable. This means that if a kernel mode process executes for an extended period of time, other processes (in particular the interactive graphics process) will have their execution delayed. Two routines within UNIX, *"copyseg(...)"* and *"clearseg(...)"*, have the potential for consuming relatively large amounts of cpu time. These routines are used to move and clear, respectively, blocks of memory in the user's address space. Since the PDP-11 instruction set does not provide an efficient *block move* instruction, these routines are relatively slow (approximately 1 ms per 512 byte memory block). Both of these routines are called quite often within UNIX in order to create new processes and to dynamically manage stack and data segments in existing processes. If a relatively large amount of memory space is involved, the execution of these two routines interferes with the interactive graphics process and ultimately causes undesirable effects on the displayed picture.

The solution to this last scheduling problem is to introduce a new concept into UNIX — that of an "interruptable" kernel mode process. *(Interruptable* in terms of allowing a process context switch to take place, even when the processor is executing in kernel mode.) This concept was implemented in a non-general way (i.e. only *copyseg(...)* and *clearseg(...)* were involved), requiring few new lines of code (70 in C and 35 in assembler), and yet satisfactorily addresses the aforementioned problem. Our installation can now support an interactive graphics process in an otherwise busy system with no degradation of performance to the graphics display (see figure 2).

Some other important features have been added to our Picture System 2/UNIX system to improve system performance in areas other than those previously mentioned. While these other features were added specifically to improve performance with respect to our graphics display, they are fairly general purpose and not limited only to graphics applications. They can be categorized as improvements in I/O throughput and program size limitations.

Normal bulk input/output data transfers in the UNIX environment fall within one of two categories: the standard, system buffered, record length independent, file access method, and an alternative "raw" I/O access method. Raw I/O is normally reserved for special usage of a standard device (such as reading a foreign magnetic tape) or to bypass the standard buffering mechanism for a special device (such as the PS2 display system). Normal access to the UNIX file system is always through a system buffer, leaving the details of mapping of logical file blocks to physical disk blocks entirely at the hands of the operating system. The disadvantage of this technique is a reduction in I/O bandwidth compared to the speeds the hardware is capable of.

In order quickly to access large amounts of data for new picture displays, a simple user interface was developed to facilitate large contiguous data transfer to and from disk storage using the "raw" I/O access mechanism. This consists of providing a set of user routines for the management of disk storage within a specified area on the standard system disk, and the inclusion of a special "indirect" device driver module to both "re-map" physical disk address translations and provide for exclusive use of the special disk areas assigned for this purpose. This technique provides for both direct data transfer to/from a user buffer (thus avoiding the cpu time required to copy the data from a system buffer) and allows for large multiblock transfers to/from disk, minimizing head seeking and per-block interrupt processing time and allowing transfers to proceed at full hardware speed.

The user subroutines provide functions very similar to the standard UNIX *open(), read(), seek(),* and

write() primitives and maintain a least-recently-used buffer pool. Both the total size and the block transfer size of the buffer pool are specified by the user during initialization. On our system disk we have allocated ten special "bulk storage" data areas, each four cylinders in size. This provides 1.2 Mbytes of storage for each area, with an average access time of 18 ms (assuming the disk is not busy processing other requests) and a data transfer rate of 1.2 Mbytes per second. Since user programs require little modification to use the new I/O routines, it is not necessary to design features into new programs in order to take advantage of the fast access bulk storage files. Programs can be originally written to use the standard UNIX file access primitives and upgraded to the new method only after it is determined that I/O throughput is too slow. Currently, *MIDS* and some movie making routines are the only programs utilizing the bulk storage data files.

[It is noted in passing that special considerations must be given to Unibus I/O transfer rates on the PDP-11/70 computer. This model processor is substantially different than other members of the PDP-11 family in the architectural design of the Unibus to main memory interface and is not capable of aggregate transfer rates greater than 1.0 Mbytes per second.[15] In addition, direct memory access output transfers over the Unibus update the contents of high speed cache memory ("cache wiping") and thus slow the execution of programs. Extensive measurements have been performed on both of these, but their presentation is beyond the scope of this paper. Note that the above comments apply only to the PDP-11/70 Unibus and that I/O transfers via the Massbus do not have the same limitations.]

Another hardware enhancement to speed overall system throughput has been the addition of an "extended core storage" device. This actually consists of 1 Mbyte of 10 μsec access bulk core memory configured to appear to both the processor and UNIX as a fixed head disk drive. This device nicely fills the gap in memory system hierarchy between main memory and moving head disk storage. It is used to store frequently accessed system data (the "root" file system and program temporary files) and has been conveniently integrated into the rest of the system. This includes some trivial modifications to account for this device's very fast access time and thus avoid the normal buffer caching mechanism present in UNIX. The overall result is faster system response time and reduced impact when the high performance graphics display is in use.

A major limitation to 16 bit minicomputer architecture is in the amount of memory directly addressable by a program at any one instant. On the PDP-11 series of computers this is generally a total of 64 Kbytes, but the memory management unit in the PDP-11/45 and 11/70 has the ability to separate programs into instruction segments and data segments, each segment capable of directly referencing 64K bytes of memory.

Four PDP-11 instructions facilitate program communication between different addressing modes and instruction/data areas in memory. These are "move to/from previous instruction/data memory space" *(mtpi, mfpi, mtpd* and *mfpd).*

Unfortunately, because of DEC's desire to "... preserve the integrity of proprietary programs"[17], the *mfpi* instruction does not function correctly when a user is attempting to access data in his own instruction segment of memory with "separated I&D" mode active. Because of the details of C subroutine calling linkage, this makes it impossible for a user subroutine to determine the actual number of arguments passed when optional arguments are possible. This is not a problem when separated I&D mode is not active, and a standard routine *(nargs())* can be called to determine the number of arguments passed to the called subroutine (of course the addressable program memory space is limited to the usual total of 64K bytes in this case).

There are a number of solutions to this deficiency that still allow programs to utilize separated I&D space, among these are:

(i) Insist that the user supply as the first argument in a subroutine call the actual number of real arguments being passed.

(ii) Implement a new UNIX system call to execute the *mpfi* instruction. (The instruction functions correctly when executed in kernel mode and is used extensively by UNIX to fetch and store data between user programs and the operating system.)

(iii) Modify the C compiler to generate an argument count on every subroutine call.

(iv) Modify UNIX to use the supervisor mode memory management registers in a way different to that which is done currently.

(v) Modify the cpu hardware to work more "correctly".

The first approach was straightforward but quickly discarded since it meant both introducing a major incompatibility between the Evans & Sutherland and UNIX version of the graphics software package (E&S does not support programs with separated I&D segments) and also the potential for user programming errors should the incorrect number of arguments be

specified.

The second approach was easiest to implement but proved unacceptable because of the overhead associated with system calls;[16] each UNIX system call takes, as previously noted, a minimum of 320 microseconds of cpu time (for a PDP-11/70 with cache) and the *nargs()* routine has the potential for making at least two, and often three or more, of these time consuming calls. In our real-time environment this approach was prohibitively slow and had to be abandoned. The third and fourth approaches were unattractive since they meant sacrificing software compatibility and standardization with other installations.

The approach finally decided upon also turned out to be rather trivial to implement once the proper section of logic was located in the central processor. The modification takes about 15 minutes to perform and involves simply cutting one printed circuit foil "etch" and adding a single jumper wire to the PDP-11/70 memory management controller (see figure 3). This change allows the *mfpi* instruction to execute correctly under all conditions, and an addition to the original *nargs()* routine that utilizes this "new" instruction when a program is running in separated I&D mode removes any previous restrictions associated with subroutines which have a variable number of arguments. This facility is used extensively by many of the real-time graphic display programs which typically have both a large program instruction segment and a large "display list" data segment and would normally not fit within the confined addressing space of a 16 bit minicomputer.

Conclusion:

The University of California, San Francisco Computer Graphics Laboratory is a relatively young facility (2½ years old), which began its mission by "building from the ground up" with both new, unproven hardware (we were the first to receive delivery of a Picture System 2 in the United States) and previously non-existent application programs in the new field of interactive computer graphics for drug design. Despite this beginning disadvantage, we have enjoyed a remarkable degree of success and are using the system on a daily production basis. Much of this can be attributed to the excellent environment provided by the UNIX operating system and the C programming language. The relatively simple modifications and extensions to the standard UNIX system provide the additional efficiency required without sacrificing the simplicity and elegance that have become a trademark of UNIX philosophy. The entire complement of high performance hardware, provided by over a dozen different vendors, has been very reliable and contributed much to the level of success. Emphasis has been placed on ease of use of interactive three-dimensional computer graphics as a powerful tool in molecular modeling and drug design and the resulting friendly environment is enjoyed by many individuals having no previous experience with computers. The system is already in extensive use at a variety of scientific research levels and popularity is growing steadily. This in turn is making increased demands for new program development, development that can proceed in parallel with production use because of the time-sharing environment provided by UNIX. This has become more than just a desirable mode of operation, it is now a necessity.

Acknowledgements:

Work supported in part by National Institutes of Health, Division of Research Resources Grant No. RR-1081. We are grateful to A.I. Wasserman for his many helpful criticisms and to J.E. Apodaca for her help in preparing the manuscript and figures. The text was prepared entirely on the UNIX time-sharing system utilizing software from Bell Laboratories, University of California, Berkeley and University of Toronto.

References:

(1) Feldmann, R.J. The Design of Computing Systems for Molecular Modeling. Annual Review of Biophysics and Bioengineering *5,* 477-510 (1976).

(2) Langridge, R. Interactive Three-Dimensional Computer Graphics in Molecular Biology. In *Computers in Life Science Research,* Plenum Publishing Corp., pp. 53-59 (1975).

(3) Levinthal, C. Molecular Model-Building by Computer. Sci. Amer. *214,* 42-52 (1966).

(4) Picture System 2/PDP-11 Reference Manual Evans & Sutherland Corporation, P.O. Box 8700, Salt Lake City, Utah 84108, Document #901130-001 A1, November 1977.

(5) Ritchie, D.M. and Thompson, K. The UNIX Time-Sharing System. Comm. of the ACM 17(7): 365-375, July 1974.

(6) Luderer, G.W.R., Maranzano, J.F. and Tague, B.A. The UNIX Operating System as a Base for

Applications. Bell System Technical Journal 57(6): 2201-2207, July-August 1978.

(7) Kernighan, B.W. and Ritchie, D.M. *The C Programming Language.* Bell Laboratories, Murray Hill, New Jersey. Prentice-Hill, Inc., Englewood Cliff, New Jersey, 1978.

(8) The UNIX Picture System 2 Graphics Subroutine Package is available free of charge to anyone possessing a valid license from Western Electric for the UNIX Time-Sharing System. To arrange for distribution please contact the authors.

(9) Newman, W.M. and Sproull, R.F. *Principles of Interactive Computer Graphics.* Second Edition, McGraw-Hill, New York, pp. 79-90 (1979).

(10) Ferrin, T.E. Write-Back to Memory Subroutine Description, UNIX/PS2 Graphics Subroutine Package, Internal Memo, 1978.

(11) Picture System 2 User's Manual. Evans & Sutherland Corporation, P.O. Box 8700, Salt Lake City, Utah 84108, Document #901129-001 NC, May 1977.

(12) Ferrin, T.E., Pensak, M. and Huang, C. MIDS: The Molecular Interactive Display System User's Manual, Version 2.6. Internal Memorandum, University of California, San Francisco, 21 July 1978.

(13) Jones, O.E. Protein Interactive Graphics User's Manual. Internal Memorandum, University of California, San Francisco, 19 July 1978.

(14) Jones, O.E., Hack, P. and Beutel, T. BILD - A Tutorial Introduction; Three-Dimensional Picture Drawing by Computer. Internal Memorandum, University of California, San Francisco, 7 July 1978.

(15) PDP-11/70 Architectural Description. Internal Technical Memorandum, Digital Equipment Corporation, Maynard, Massachusetts 01730, 1977.

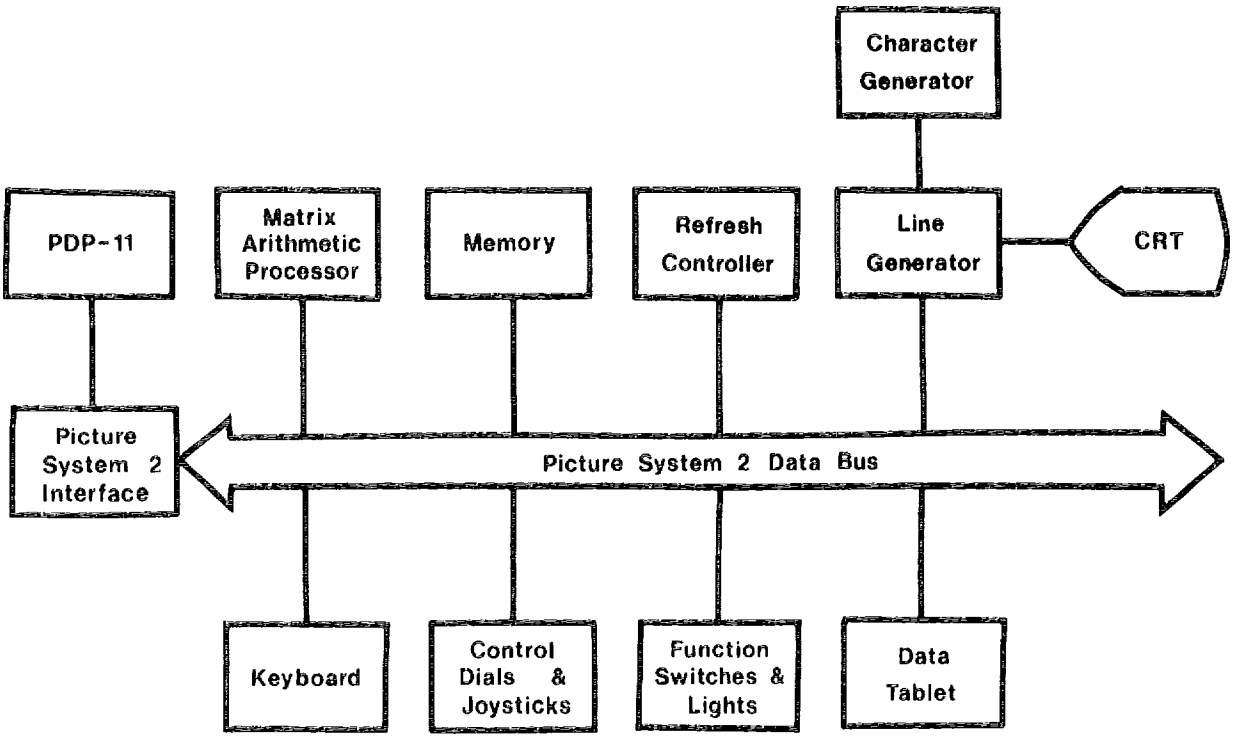
(16) A solution to reducing potential system call overhead associated with the *nargs()* subroutine has been proposed in a personal communication with Tom Duff from the New York Institute of Technology. This involves keeping a most recently accessed list of program counter values and the associated number of arguments to the called subroutine, the latter determined through a system call. The next call to the same subroutine from the same program location can then avoid the system call and quickly determine the number of subroutine arguments by searching the above list.

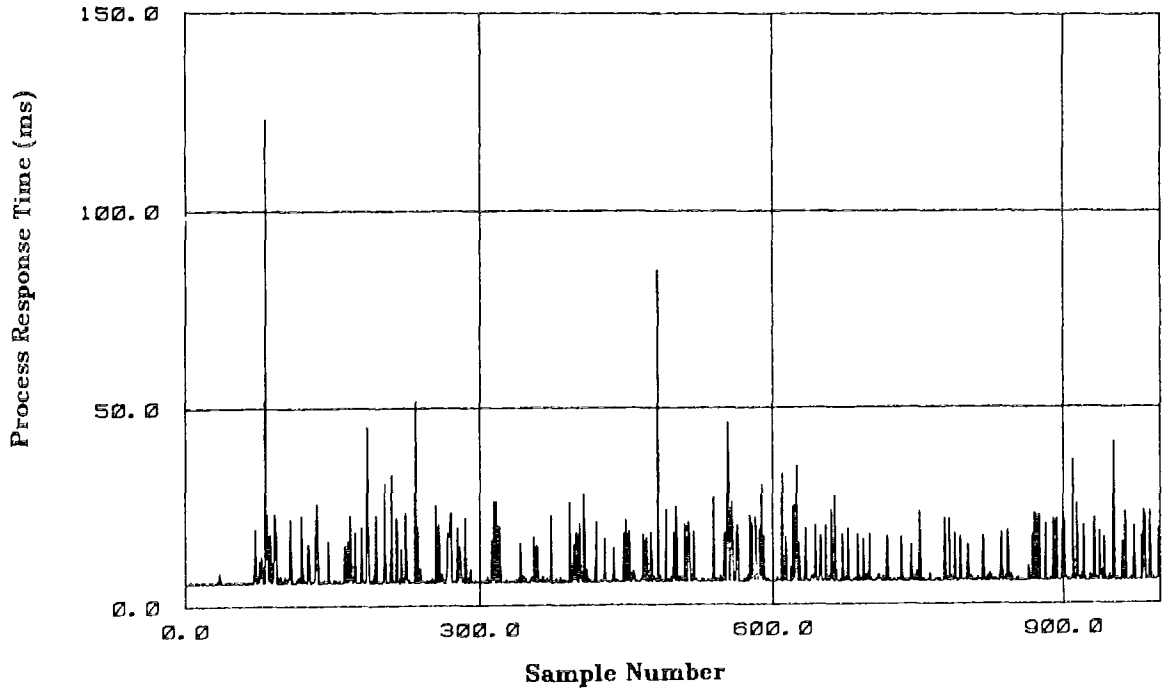
(17) KB11-C Processor Manual (PDP-11/70). Digital Equipment Corporation, Maynard, Massachusetts 01730, Document #EK-KB11C-TM-001, pg. IV-3-5, 1975.

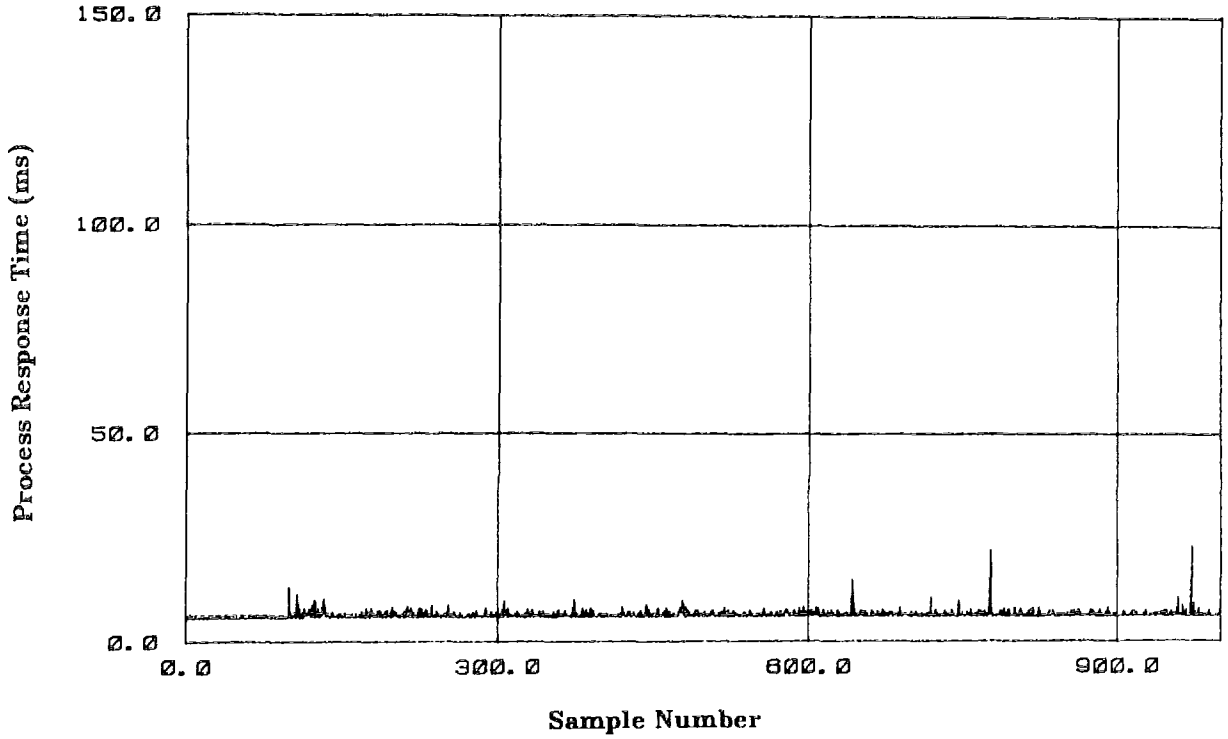
Figure 1: **Picture System 2 schematic diagram.** Some components, most notably the PS2 interface and matrix arithmetic processor, have the ability to function in either an active or passive mode of operation and thus provide overall system flexibility.

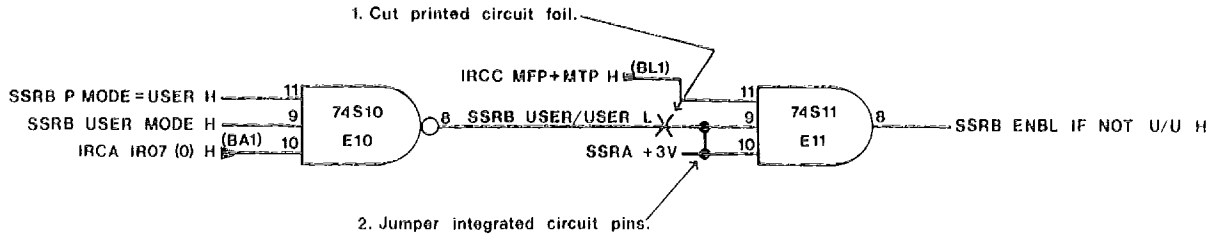
Figure 2: **Process response time.** Shown is the response time for a high priority process running in a multiuser UNIX environment. In the top figure the process priority was controlled using the standard UNIX *nice()* system call to adjust the process' priority to a low value and therefore schedule its execution before other users. In the bottom figure the process used the newly added *rtp()* system call to guarantee it the quickest possible response time. (Courtesy of M. Wallen, University of California, San Diego).

Figure 3: **Modifications for *mpfi* instruction.** A special circuit in the PDP-11/70 memory management control unit (module # M8138-YA, "System Status Register (SSRB)") was defeated to allow more logically consistent instruction execution.




150. 0
100. 0
50. 0
0. 0
Process Response Time (ms)
0. 0
300. 0
600. 0
900. 0
Sample Number


150. 0
100. 0
50. 0
0. 0
Process Response Time (ms)
0. 0
300. 0
600. 0
900. 0
Sample Number


1. Cut printed circuit foil.
IRCC MFP+MTP H
(BL1)
SSRB P MODE = USER H
SSRB USER MODE H
IRCA IR07 (0) H
(BA1)
11
9
10
74S10
E10
8
SSRB USER/USER L
SSRA +3V
11
9
10
74S11
E11
8
SSRB ENBL IF NOT U/U H
2. Jumper integrated circuit pins.