SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 12, 1133–1145 (1982)

An Overview of the Yale Gem System

JOHN LEVINE
*Department of Computer Science, Yale University, New Haven, Conn. 0652, U.S.A.**

SUMMARY

The Gem system is an experimental computing facility that provides low cost, high speed, graphics-oriented computing to between ten and sixteen simultaneous users. It provides many unusual facilities to its users and presents a user interface that is unique in its convenience and flexibility. The motivation for the system, its design and user experience are described. Possible future avenues of research are also outlined.

KEY WORDS Graphics Editors Time-sharing Interactive systems CRT Terminals

MOTIVATION

In 1972 the Yale Computer Science department undertook to build an inexpensive computing system for student use. The goals were to provide interactive computing to each user, and to allow the use of text and graphical output in any combination. Since the system had to be inexpensive, substantial effort was made to build it out of standard commercially available components whenever possible. Also, since the project would evolve rapidly over the years, it was imperative to maintain the maximum of flexibility to allow for reconfiguring the system for unforeseen uses.

HARDWARE DESIGN

The Gem system hardware is pictured in Figure 1.

Display system

The heart of the system is the Gem memory, a bank of 256K 16-bit words of semiconductor memory, which is of course called Gemory. Each screen is assigned a 16K (16 bits/word) section of Gemory. All the screens are refreshed simultaneously from their respective Gemories 30 times per second.

The image on each screen is considered to be an array of bits 576 wide and 454 high. These numbers come quite naturally from the resolution of the standard television monitors used and the size of the screen memory. The most common way we use the screen is to treat it as lines of text. Each character is eight bits wide, since our computers address 8-bit bytes, so the screen allows 72 characters across. The 16K words of screen memory are just about filled up by 454 such scan lines, and the aspect ratio is close to the natural 3/4 ratio of the screens.

0038–0644/82/121133–13$01.30 *Received 11 October 1979*
 Revised 25 May 1980

* Present address: Interactive Systems (Calif.), P.O. Box 349, Cambridge MA 02238, U.S.A.

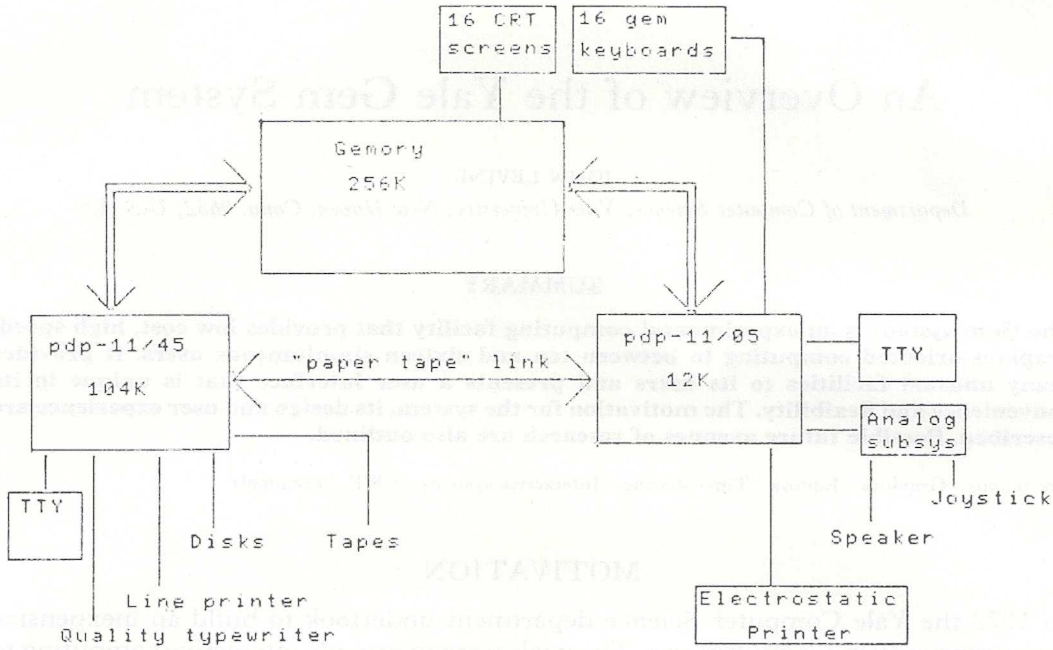

Figure 1. Block diagram of the GEM system. There is a good deal of conventional computing equipment here not referred to in the text. As was mentioned above, the program used to draw this was developed in less than an hour

Each 36 word section of memory corresponds to one scan line on the screen, with the first 36 words corresponding to the top line on the screen. The low order bit of the first word in a group is the leftmost dot on the line, the next bit in that word the next dot to the right, and so on. A '1' bit is a lit spot on the screen; a '0' bit is dark spot. Usually we draw white figures on a dark background because it looks better on the screen, although the hardware has no natural preference either way. (The printer on which the figures for this article were drawn prints black on white, the opposite of the screen convention.)

Pictures are drawn on the screens by the simple act of writing bit patterns into Gemory. Changing individual words or bytes of memory changes individual sections of the screen. Operations such as blanking out a selected section of the screen or copying an image from one part of the screen to another part of the screen are easily programmed.

Two computers access the Gemory. The 'terminal' computer, which is a Digital PDP-11/05, has the primary job of simulating more or less conventional typewriter terminals. The other one, a PDP-11/45, does general-purpose time-sharing. Each computer has mapping hardware that allows it to access the Gemory for any terminal as though the Gemory were part of the primary memory for the computer. It is quite possible for both computers to access the same screen memory at the same time; hardware resolves the contention. The primary connection between the two computers (other than through the shared Gemory) is a simple bidirectional link that appears to each machine to be a very fast paper tape reader and punch. The terminal computer has been programmed to send the characters typed on the keyboards (except for locally handled functions) to the main computer, and the main computer sends characters to be displayed to the terminal computer.

Other hardware

Several peripherals are attached to the two computers, a few of which bear further discussion. A chronic problem facing CRT oriented systems is the difficulty of obtaining paper copies of the contents of the screen at reasonable cost, especially when the screen can display pictures as well as text. The terminal computer has attached to it a standard electrostatic matrix printer. Since the data in a screen memory is an array of bits in the same format as is required by the matrix printer, copying the screen memory directly to the printer produces a paper copy of the screen image. We have programmed the terminal computer so that on each keyboard, there is a 'Print' button that makes a copy of the screen, allowing users to obtain hard copy as often as needed. A user program can also arrange to copy screen images to the printer by sending control character sequences to the terminal, so that by repetitively drawing pictures and printing them longer paper pictures can be drawn. All sixteen terminals share the same printer, so that the cost per terminal of providing the printer is quite low. All of the figures for this paper were printed on the matrix printer.

There is also an analogue I/O subsystem, which is described later, attached to the terminal computer.

SYSTEM SOFTWARE

After a good deal of investigation and experiment, we decided to adopt the well-known Unix time sharing system.[1,2]

Terminal computer

We have programmed the terminal computer to emulate an ASCII CRT terminal. The emulation program is written in the same C language [3] as is the Unix system itself and so the program has been easy to develop and modify.

Each Gem terminal can be used just like a typical typewriter terminal. Since the behaviour of the terminal depends solely on the program in the terminal computer, we have been able to develop a terminal with some very unusual characteristics. In fact, changing terminal characteristics has been so easy that when we receive programs written elsewhere that depend on the special features of a particular type of terminal, it has often been easier to reprogram the terminal computer to emulate the desired kind of terminal than to change the programs themselves.

We have the usual control functions implemented: cursor motion, cursor home, cursor addressing, horizontal tabulate, ring the 'bell', carriage return, line feed (possibly scrolling at the bottom of the screen,) clear page, clear line, reset, and scroll the screen image any distance up or down. There are also some more unusual features. For example, since scrolling the screen image is a very slow operation, the user can, under program control, set the distance to be scrolled when he sends a line feed at the bottom of the screen. We have discovered that for most purposes scrolling 10 lines every tenth line feed is perfectly adequate and puts a much lighter load on the terminal computer than does conventional scrolling. The user can at any time set conventional line-at-a-time scrolling if wanted.

Several modes can be set in the terminal, either by pushing buttons on the keyboard or by sending control character sequences from the main computer. The terminal can be switched to use the APL character fount rather than the regular 96 character ASCII set, and characters from the two can be intermixed, since text already written does not change when the terminal mode is changed. One can also switch between having characters overstrike as on a real typewriter and having only the most recent character at a screen position show, as on a conventional CRT. For demonstrations, the terminal can switch to double size characters.

To help exploit the unique features of CRT terminals, we have implemented screen 'windows' which allow screen activity to be restricted to an arbitrary sub-rectangle of the screen. A program can send a control sequence that sets the upper, lower, left and right margins of the screen window. All of the regular terminal-control characters such as cursor motion, clear page, horizontal tabulate, and scroll up or down then operate only within the selected window. Unlike most windowing systems, no program in the main computer need be aware that it is writing to a window rather than to the full screen, since the terminal computer handles all such activity automatically.

Also note that functions like 'clear to end of page' are easily simulated by performing an existing screen operation within a window. (In this case, the function to use would be clear-page.)

Communication between the terminal computer and the main computer uses an extremely simple protocol. The link between the machines transmits 16 bit words of data in parallel. Each word transmitted in either direction has the logical terminal number in the high byte and the character to transfer in the lower byte. Each character is fully processed at the receiving end before the next character is received. This avoids having any start/stop protocol. In fact, early versions of the terminal emulator program did have a start/stop protocol that buffered characters in the terminal computer but we discovered that the amount of computing needed in the terminal computer to implement the protocol was great enough that we got better response by processing each character to completion immediately. The main computer sends characters in round-robin fashion for the various terminals so that each user gets a fair share of the terminal computer's time.

The program inside the terminal computer is also very simple in concept. It consists of a loop that obtains a character and terminal number from the main computer or from an internal buffer used for 'local' functions, sets the Gem registers to map the screen memory for the required terminal, and then either draws printable characters directly or dispatches to routines for interpreting control characters.

For each screen it maintains a small table containing the cursor position, screen window limits and other related parameters that are updated after each character or screen operation.

Despite all of the advanced features in our terminals, the emulator program only takes about 6500 words of memory (not counting about 3000 words for the character tables).

Operating system

Minimal modification of the operating system was needed in order to exploit the full power of the Gem terminals.

We added a mechanism to the operating system that lets any program manipulate the screen memory of its terminal. We have exploited some otherwise unused mode-switching hardware in the PDP-11 to provide a direct hardware path from every running program to the section of Gemory associated with its terminal.

The PDP-11/45 has three operating modes, each with a full set of address mapping registers. There are also 'previous address space' instructions that let a program access the address space of a different mode than the one in which the program is running. The Unix system only uses two of the three available modes, so we modified it so that the screen memory of the controlling terminal of a process was always mapped into the address space of the hitherto unused mode and that the 'previous mode' for a user program was always that mode. Once the registers are set up at system initialization time, this feature adds only one instruction to a context switch, an insignificant slowdown.

The overall effect is that a user program need only execute some special instructions, and words of data are moved in and out of the screen memory without any further intervention from the operating system. A screen memory can, in this way, be totally rewritten in about 50 ms, which looks like an instantaneous change of the picture. A user can also map the screen memory directly into the address space of a running program, which provides faster screen access but severely limits the space available for the program itself, since the screen takes 16K of the available 32K program address space. In fact, most programs use the first access mechanism since it is usually fast enough.

Application routines

All of the major languages used on the system, including Fortran, Basic, APL and C, have had some sort of graphics interface added. The most sophisticated routines, which are described below, are for programs written in the language C.

A variety of user software has been written to assist in picture creation. The most widely used package lets the user draw lines and points in subrectangles of the screen. A frequent approach is to define several screen windows, some for graphics, some for text and some for a combination of both. For example, there is a screen editor which has a large window which shows a portion of the file being edited, and a small one-line window which holds arguments to editor commands.

Another package allows character founts to be created, edited and used to draw text and pictures on the screen. Besides a wide variety of character sets such as regular Roman, Greek, Cyrillic and old German, there are character sets that are used to build more complicated pictures such as musical notes and map symbols, and various novelty characters including excellent reproductions of the signatures of some of the faculty members.

Other less widely used software includes Calcomp-compatible Fortran routines for drawing lines and characters, and some routines for viewing three-dimensional objects in perspective from a variety of viewpoints.

APPLICATIONS

Editors

By far the most heavily used piece of graphics software is the screen oriented text editor, which is based on Yale's extensive previous work on screen editors.[4] On many other systems, screen editors have failed to gain acceptance because they have verbose and confusing syntax and because they generally place a severe load on the computing system or else run very slowly. The approach we use is that the editor provides a

```
.sp
1
Block diagram of the GEM system.
There is a good deal of conventional computing equipment here not
referred to in the text.
As was mentioned above, the program used to draw this was developed
in less than an hour.
.sp
2
Sample editor session.
This example shows the words 'display arbitrary polyhedra' selected
for an operation.
.sp
3.
A graphics oriented game.
Note the use of a text window on the right to display instructions.
This window is also used for error messages and comments from the
program.
.sp
4.
A graphics tree editor.
This particular editor was written as an assignment for a class.
The symbols are from a standard font and do not, in this case,
mean anything.
.sp
5
Display from a curve plotting program.
Note the ability to easily plot a great deal of information.
(This data is not from the Gem system.)
.CH References
.PR

** cursor defined **                    File gem2.n Line 231
```

Figure 2. Sample editor session. This example shows the words 'display arbitrary polyhedra' selected for an operation

window into the file, and that the user can simply cross out and overwrite much as he would on paper. Any text that the user types onto the screen goes directly into the file being edited. Using single-key commands, the user can invoke a variety of cut-and-paste operations, as well as the usual editor operations of moving forward and backward in the file and doing context searches. He can draw a box around some part of the text on the screen and then delete it, move it somewhere else in that file or to a different file, or even execute it as commands to the system command interpreter.

By careful design of the terminal emulator program and the interface between the terminal computer and the main computer, we have been able to make the screen editor run faster and cheaper than the various line editors, so that most users edit only with the screen editor and many do not even know how to use any other. Beginning users find the editor very easy to use because most of the operations are intuitively easy to grasp, and have close analogies in things they do while typing on an ordinary typewriter.

Editor implementation

Our screen editor on the Gem system was specifically designed to achieve maximum performance. The most obvious limitation of the editor is that files that are to be edited are limited to 16K characters. This turns out usually not to be a problem because most of the files that users want to edit are small anyway, and because the Unix system already has a variety of tools that make it convenient to break large files into pieces and to handle groups of files as a unit.

The editor reads the entire file to be edited into program memory, edits it 'in core' and rewrites it completely when editing is done. We added a special terminal mode to the system for editing. In this mode, characters that arrive from the keyboard are divided into two classes: cursor motion and printing characters are simply echoed back to the screen and buffered in the system, whereas other control characters are not echoed and cause all of the buffered text to be passed to the program. In the case of the characters that are echoed this way, the echo to the terminal produces a correctly updated screen image and the underlying editor program need take no immediate action. For the other characters, the editor needs to change the screen so that the editor program reads and accounts for the buffered characters and then performs the desired function. The advantage of this scheme is that the editor program need be swapped in and run only in the comparatively infrequent event that a control function is requested, and that quick response is guaranteed for cursor control and text characters which comprise most of what is typed to the editor.

In practice we find that even for the functions that require waking up the editor program, response time is usually under a second, whereas for the faster functions response seems instantaneous.

Picture creation

We have found that simple-minded graphics programs are extremely easy to write. For example, the program used to draw Figure 1 (the block diagram of the system) only took about half an hour to write and debug. The ability to avoid having a display list and to work directly with the screen image lets the user bypass a whole level of complexity inherent in most other graphics systems. If there is some need for more complicated display data structures, it is usually very easy to write the routines that translate them into the actual screen image. It is also possible, though we have not done it much, to read back the picture for further analysis. The most common application of this is that we usually draw lines in 'complement mode', i.e. inverting the bit values on the screen rather than just turning them on, which has the very useful effect that a line can be removed just by redrawing it.

The APL subsystem makes use of the APL character fount and lets line drawings be made directly from APL. There has also been work done on mapping rectangular areas on the screen directly into APL arrays so that pictures can be drawn with the full flexibility of the APL operators, working directly on the hardware representation without interposing relatively inefficient and inflexible line-drawing routines.

Some students have investigated picture creation languages, such as Logo[5] and others have created relatively sophisticated systems to manipulate and display arbitrary polyhedra in perspective with hidden line elimination. Although animation is quite difficult, display of arbitrarily complex pictures is easy since there need be no display list that grows with the complexity of the picture.

This ability has encouraged some cartographic work. There is a set of routines with which users easily create maps of the United States and display information keyed by Zip code. This is of interest to groups which maintain mailing lists on the machine. An undergraduate has created a sophisticated package which keeps track of information

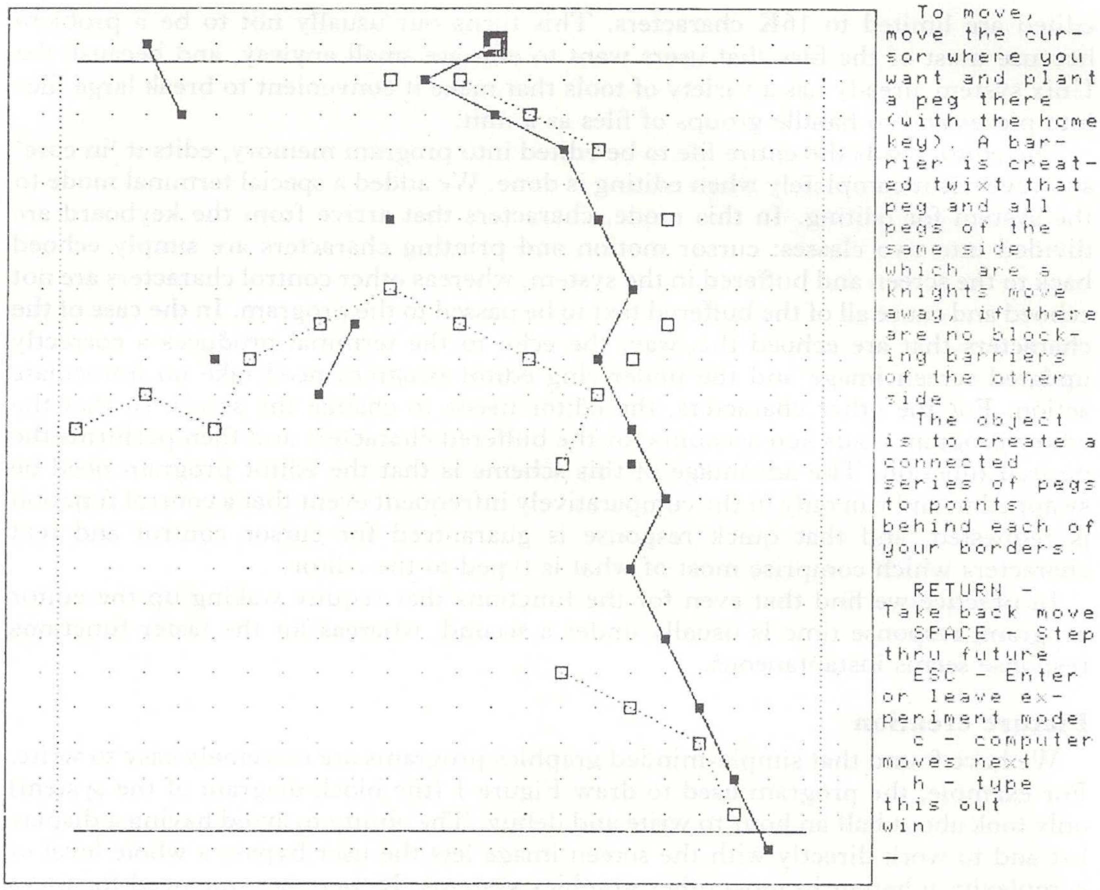

Figure 3. A graphics oriented game. Note the use of a text window on the right to display instructions. This window is also used for error messages and comments from the program

about Yale's underground utility tunnels. It stores data about which tunnels are connected to which others, what types of pipes go in which direction, which doors have locks, where the burglar alarms are, etc. The interface to this is almost entirely graphical: the user has a map of the campus on which he can overlay the various tunnels and facilities. Information is added and deleted by pointing to the building or tunnel of interest (by moving a cursor around the screen) and then typing the changes.

Analogue I/O applications

We enhanced the terminal emulator so that bytes of Gemory could be converted to and from analogue signals, using the above mentioned analogue I/O subsystem. An exciting application is the analysis of electrocardiagrams, done in co-operation with the Veterans' Administration. Analogue tapes of patients' heartbeats are read into the A-to-D converter on the terminal computer and the signal digitized every 200 μs. The digitized information stored in Gemory is then read into the main computer and usually written on computer digital tapes. People working at the terminals can then rapidly analyse these tapes. Individual heartbeats are shown graphically on the screen,

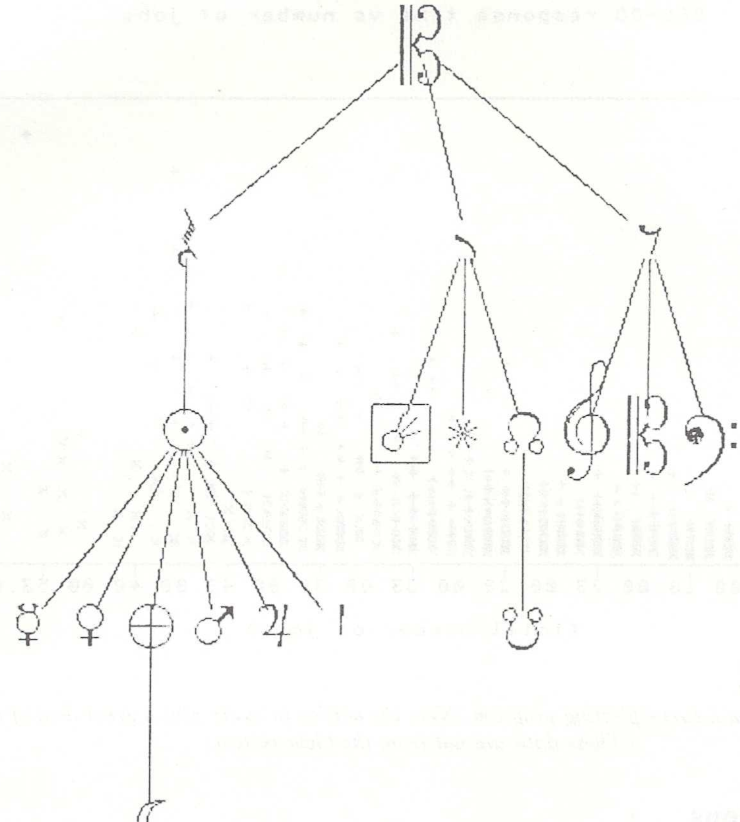

Figure 4. A graphics tree editor. This particular editor was written as an assignment for a class. The symbols are from a standard fount and do not, in this case, mean anything

and the operator characterizes each as normal or abnormal. As heartbeats are analysed, they are stored so that further similar heartbeats can be identified automatically. The entire data from a typical 12-hour tape can be thoroughly analysed in about half an hour, which is about an order of magnitude faster than any other method of comparable accuracy. The resolution of the Gem terminals and their ability to display arbitrarily complex pictures are crucial to this application, since ten or twenty heartbeats are displayed simultaneously, each with maximum screen detail.

Having noticed that analogue signals could be read into the Gem system, some students investigated the feasibility of producing output analogue signals in real time, and thus a system that plays music was written. It can play six to ten voices simultaneously, and operates reasonably well even when other users continue working, which is unusual for computerized music synthesis. Work is now in progress to develop a music score editor that lets users manipulate music in the conventional musical notation and to integrate it with the music playing system.

The analogue input system has also been used to handle a joystick for picture drawing. We found that for many purposes, cursor keys on the keyboard are more convenient for pointing than the joystick is, so there has been comparatively little use of it so far.

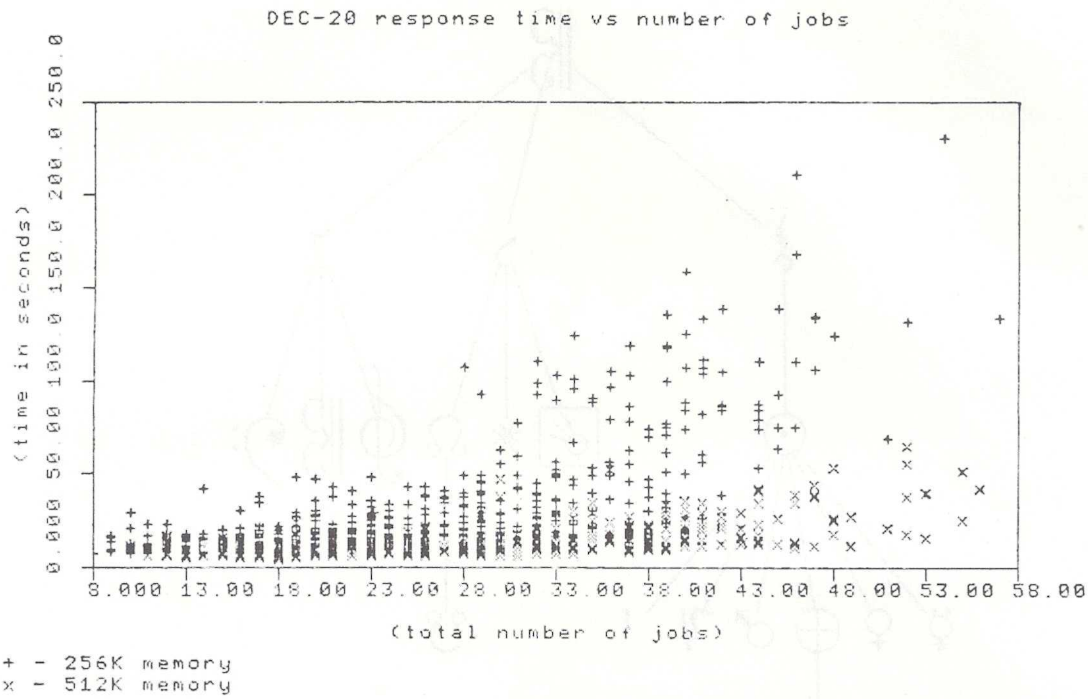

Figure 5. Display from a curve plotting program. Note the ability to easily plot a great deal of information. (These data are not from the Gem system)

Other applications

Naturally, a wide variety of games and demonstrations rapidly appeared. The usual time-of-day command has been supplemented by a 'clock' command which draws a clock face on the screen, with the hands indicating the correct time. A sweep second hand is optional. When a terminal is idle, the system displays a picture from a library of appropriate messages, such as the seal of Yale University, various portentious mottoes, and other computer artwork. This actually had some practical benefit, since users can more easily identify available terminals and, since having a picture selected for the library is considered something of an honour, there was an incentive to develop some of the picture display packages.

Two undergraduates developed a 'Star Wars' game which lets several people at different terminals fly space ships, land on planets, and of course blow each other to smithereens. Each person sees the universe out of the windows of his own ship, and has a set of controls and indicators. The illusion of being in a 3-D space is quite persuasive, and the entire game is very involving.

USER REACTIONS

The Gem system provides a user interface that is quite different from that provided by any other time-shared computing system of similar cost that we know. It is one of the only systems to provide screen-oriented editing (as opposed to typewriter-oriented editing adapted to a CRT terminal). It is absolutely the only system that allows ordinary users at any terminal to do graphics without making special arrangements in

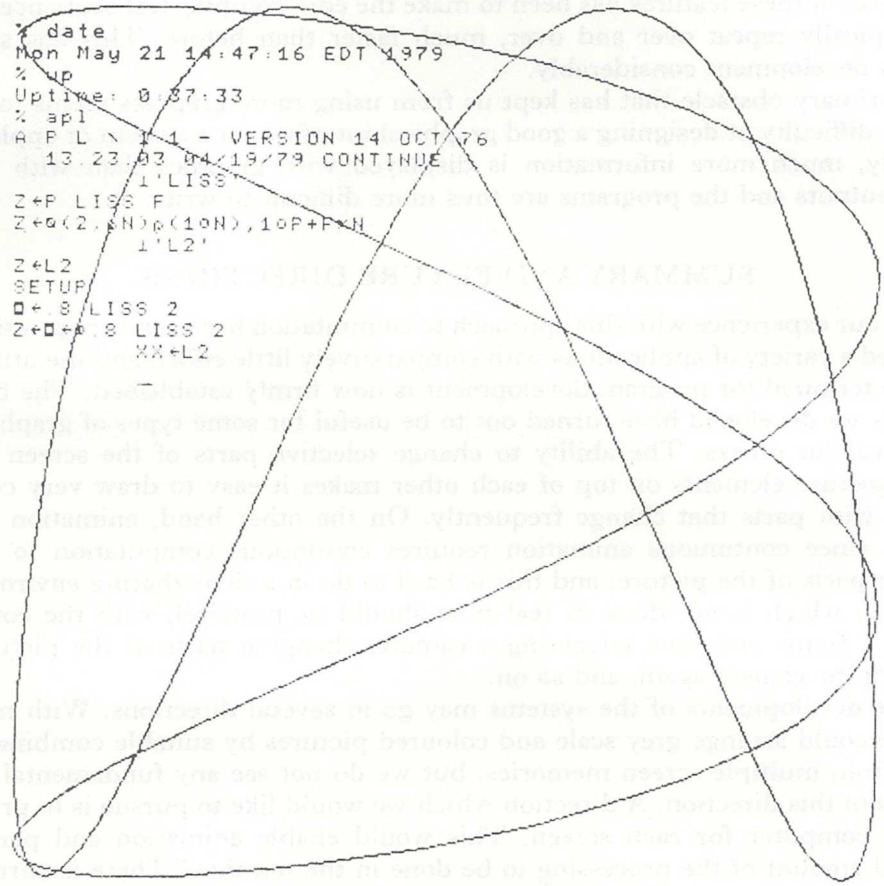

Figure 6. Output from a small APL program. Note the combination of regular characters at the top with APL characters in the middle and the graphical figures

advance and having to use special terminals different from the ones used for normal work. Every introductory computing course that uses the Gem system includes a few assignments involving graphics. Some of them have been surprisingly sophisticated, as for instance one that simulated the spread of pollutants downwind from an explosion.

All users use the graphics editor for text and program preparation. It is much easier to learn than the standard line editor and, for most functions, allows faster editing. Our experience has been that when users move to other systems that do not provide window editing they have reactions not unlike those of people accustomed to interactive computing who have to use batch systems.

The editor has recently been augmented so that a user can automatically execute a system command of his choice upon leaving the editor, typically to compile a program or reformat a document. Another increasingly popular feature lets the user execute part or all of a file as system commands. The commands can involve the use of variables, conditional statements and branches. Some users now have files of favourite commands from which they select pieces to execute.

An effect of these features has been to make the edit–compile–test sequence, which users typically repeat over and over, much faster than before. This has sped up program development considerably.

The primary obstacle that has kept us from using more graphics seems to be the inherent difficulty of designing a good graphical interface for a system or application. Typically, much more information is displayed with graphics than with regular textual outputs and the programs are thus more difficult to write.

SUMMARY AND FUTURE DIRECTIONS

Overall, our experience with this approach to computation has been very positive. We developed a variety of applications with comparatively little effort, and the utility of a graphics terminal for program development is now firmly established. The bit-map terminals we developed have turned out to be useful for some types of graphics and impractical for others. The ability to change selective parts of the screen and to overlay picture elements on top of each other makes it easy to draw very complex pictures with parts that change frequently. On the other hand, animation is very difficult, since continuous animation requires continuous computation to redraw changing parts of the picture, and this is hard to do in a time-sharing environment. Animation which is not done in real time should be practical, with the computer drawing a frame and then triggering a camera, changing parts of the picture and triggering the camera again, and so on.

Future developments of the systems may go in several directions. With minimal effort we could arrange grey scale and coloured pictures by suitable combination of images from multiple screen memories, but we do not see any fundamental breakthroughs in this direction. A direction which we would like to pursue is to provide a terminal computer for each screen. This would enable animation and permit an increased amount of the processing to be done in the terminal. There is currently a restriction that all terminals must be in the same building as the computers, since the screen images are now transmitted via coaxial cables which are impractical over long distances; the individual terminal computers would alleviate this. Our work on windows points the way to transmitting complex pictures with minimum transmission time, by transmitting only the minimum of windows needed to update or maintain a screen image. There is also opportunity for work on graphical input devices, such as tablets and mice.

We also plan to do further work in integrated graphical environments, extending the screen windows to be more generally useful, with different programs simultaneously accessing different windows on the same screen, somewhat in the manner of Teitelman's 'Programmer's Assistant'[6] or the IBM 3270 Session Manager.[7]

Since the screen editor seems to be so generally useful, we are moving toward making it the standard system interface, so that users only leave it occasionally to do something unusual. Programs can be run directly from the editor, and their input and output data can be manipulated just like any other file. In some cases, the output from a program would even be edited and then fed back into the same program for further processing without leaving the editor; this is useful in word-processing applications.

Our bit-map terminals make it easy to draw characters on the screen from a variety of different founts. This would allow technical and scientific reports which include mathematical symbols and letters from foreign alphabets to be typed up directly.

(Current systems for this purpose require that codes for the symbols be used which are only translated to the correct form when the document is finally printed.) We could then provide a system that let such documents be prepared with full visual fidelity maintained from initial keying through editing to final printing. Such a system would be equally useful for producing slides and transparencies. A matrix printer with higher resolution than the one we now have would be needed to print documents of acceptable quality for distribution.

We believe that much work remains to be done on graphical programming tools. We are now developing a screen-oriented program editor which recognizes the syntax of the programming language, so that editing commands can be phrased in terms of the language of the program rather than just in terms of lines and characters.

Finally, many of us have noticed that a graphics terminal allows a program to put an immense amount of information on the screen in a very short time. Large portions of most graphical programs are dedicated to maintaining the data structures that hold the information displayed. There has been interest in creating data base packages tailored to the graphics environment so that complicated pictures can be more easily manipulated. The fount routines mentioned earlier are a simple example of this.

ACKNOWLEDGEMENTS

The original conception of the Gem system was due to Edgar T. Irons and Peter Weiner. The terminal hardware was primarily designed and built by Charles Minter and Mark Brown. The keyboard subsystem was designed and built by Inder Singh. Robert W. Tuttle is responsible for the design of the two-computer system and also wrote the screen editor. John Levine designed and wrote the screen access features of the operating system and the terminal emulator program. The electrocardiogram software was designed and written by John W. Lewis. The music software is due to Steven M. Haflich, of the Yale Department of Music.

This work was funded in part by grants from the Sloan Foundation and the Exxon Foundation.

REFERENCES

1. D. M. Ritchie and K. Thompson, 'The UNIX time-sharing system; *CACM* **17** (7), 365–375 (1974).
2. *Bell System Technical Journal,* **57** (6), part 2, 1897–2312 (1978).
3. B. W. Kernighan and D. M. Ritchie, *The C Programming Language,* Prentice-Hall, Englewood Cliffs, 1978.
4. E. T. Irons and F. M. Djorup, 'A CRT editing system', *CACM,* **15** (1), 16–20 (1972).
5. S. Papert, 'Teaching children thinking', *IFIP Conference on Computing Education,* North-Holland, Amsterdam, 1970.
6. W. Teitelman, *The Programmer's Assistant,* Report CSL 77–3, Xerox Palo Alto Research Center, 1977.
7. J. M. McCrossin, R. P. O'Hara and L. R. Koster, 'A time-sharing display terminal session manager', *IBM Systems Journal,* **17**(3), 260–275 (1978).