message contains only one control bit, called the acknowledge bit. The operational rules for both terminals are:

1. If the previous reception was error-free, the acknowledge bit of the next transmission is one; if the reception was in error the bit is zero.
2. If the acknowledge bit of the previous reception was zero, or the previous reception was in error, retransmit the old message; otherwise fetch a new message for transmission.

The question of when to accept an error-free reception is left open. This question, in fact, has no consistent resolution. Consider the message exchanges depicted in Figures 3a and 3b. Specifically, should the message received at line 7 be accepted by A? A is presented with exactly the same information in 3a and 3b! A is forced to guess which situation is the one that has occurred. The penalty for a wrong guess is either dropping a message or accepting a duplicate of a message.

If A consistently assumes that 3a represents the situation, A will pick up message duplicates in the (rare) case when two errors occur in sequence as in 3b. Such errors, while rare, do occur, and their rareness will make it extremely difficult to catch the flaw in the system. This inadequate scheme will work *almost* all of the time.

6. Conclusion

A field-proven scheme for achieving reliable full-duplex transmission over noisy half-duplex telephone lines has been presented. The sensitivity of the algorithm and the difficulty of the problem have been illustrated by contrasting the algorithm with another, slightly different algorithm. This modified algorithm fails in rare cases and gives rise to operation which is faulty enough to degrade its usefulness, and not faulty enough to permit it to be easily debugged.

An interesting problem is posed by these two algorithms. The adequate scheme used two bits of control information (verify and alternation bits) per message while the inadequate scheme used only one bit (the acknowledge bit). In Section 3, three states were described for the received message, and the control bits of the next transmission encoded into the two control bits the total information concerning which of the three states held on reception. This leads to the conjecture that at least two control bits are required for any adequate scheme of this sort, and that only one control bit will never do. The reliable duplex transmission problem would, of course, have to be better formalized before it could be claimed that such a conjecture were "proven."

On the Design of Display Processors

T. H. Myer
Bolt Beranek and Newman Inc, Cambridge, Mass.
AND
I. E. Sutherland*
Harvard University, Cambridge, Mass.

The flexibility and power needed in the data channel for a computer display are considered. To work efficiently, such a channel must have a sufficient number of instructions that it is best understood as a small processor rather than a powerful channel. As it was found that successive improvements to the display processor design lie on a circular path, by making improvements one can return to the original simple design plus one new general purpose computer for each trip around. The degree of physical separation between display and parent computer is a key factor in display processor design.

KEY WORDS AND PHRASES: display processor design, display system, computer graphics, graphic terminal, displays, graphics, display generator, display channel, display programming, graphical interaction, remote displays
CR CATEGORIES: 2.44, 6.22, 6.29, 6.35

1. Introduction

In mid-1967 we specified a research display system. This paper describes some of the problems we encountered and some conclusions we have drawn. The display will be an adjunct to an SDS-940 time-shared computer system. The chief purpose for the display and the parent computer is programming research.

When we first approached the task, we assumed we had merely to select one of the several available commercial displays. This proved possible with the analog equipment that constitutes a *display generator*; we found several display generators that combined good accuracy, resolution, and speed. However, the control part of the display, which we have come to call the *display processor*, was another story. We were not completely happy with the command repertoire of any of the commercial systems we saw; we were not sure just how to couple the display to our computer, and above all, we had serious doubts about what a display processor should *be*.

This work was sponsored by the Advanced Research Projects Agency under ARPA Order No. 627, Amendment No. 2, and conducted under Contract No. AF19(628)-5065, Air Force Cambridge Research Laboratories, Office of Aerospace Research, United States Air Force, Bedford, Massachusetts 01730.
* And Bolt Beranek and Newman Inc, Cambridge, Mass.

Finally we decided to design the processor ourselves, because only in this way, we thought, could we obtain a truly complete display processor. We approached the task by starting with a simple scheme and adding commands and features that we felt would enhance the power of the machine. Gradually the processor became more complex. We were not disturbed by this because computer graphics, after all, *are* complex. Finally the display processor came to resemble a full-fledged computer with some special graphics features. And then a strange thing happened. We felt compelled to add to the processor a second, subsidiary processor, which, itself, began to grow in complexity. It was then that we discovered a disturbing truth. Designing a display processor can become a never-ending cyclical process. In fact, we found the process so frustrating that we have come to call it the "wheel of reincarnation." We spent a long time trapped on that wheel before we finally broke free. In the remainder of this paper we describe our experiences. We have written it in the hope that it may speed others on toward "Nirvana."

2. The Wheel of Reincarnation

The simplest displays merely plot points from coordinate information. The TX-0 display at MIT (circa 1957) or the PDP-1 with DEC Type 30 (circa 1960) are of this type. Such a display has no processor; it is tied to the central registers of the parent computer. To display a point, its coordinates are first loaded into the central registers of the computer. For example, with a DEC Type 30 and a PDP-1 the accumulator is loaded with x and the input-output register with y. A display command is then executed which results in a point flashed on the screen.

One problem with this scheme is that the processor is tied up in generating display. If an attempt is made to compute concurrently with display, the display may develop an objectionable flicker. The situation seems even worse when one considers that refreshing a static display is a repetitive operation that need not occupy an entire processor full time.

For just a little more money one can buy a data channel for the display. The data channel has a display address register and a word counter. The channel takes successive data words from a display file in core until the word count goes zero, at which point the central processor restarts the channel at the beginning of the display file. Now the processor is freed for other work and the display can operate as fast as its analog circuits permit.

Point-by-point display is, of course, expensive of time and memory, even with a data channel. Any modern display should be able to draw lines and plot characters automatically. For such a display delta x and y information and characters will appear in the display file, as well as position values. In addition, there must be codes to set intensity and to tell whether beam movement is to generate a line or a point. These codes are regarded as new kinds of data for the display.

Now someone points out that a special code to stop the channel—a channel halt—could be used to end the display file. The word counter could be eliminated, thus saving money. At this time one realizes something one had begun to suspect earlier—that a display is inherently unlike other input/output devices. A magnetic tape unit, for example, must be able to transmit arbitrary combinations of bits onto tape. The display, on the other hand, may interpret some combinations of bits in its data as special commands, since its only function is to post a picture on the screen.

For just a little more money one can add some other commands to the display data channel. One is a jump command. This allows the channel to display a file repetitively—to refresh the display without intervention from the central processor. It also provides more flexibility in handling display data, since the channel can now handle noncontiguous display files.

In many engineering applications the pictures which will be displayed have repeated subpictures such as circuit symbols or small parts. So, for just a little more money, one adds a subroutine feature to the display's data channel. Repetitive circuit symbols can now be drawn by successive calls to appropriate channel subroutines.

The subroutine feature requires two new commands and means adding a new register to the display channel. A subroutine jump command saves the return address in a special register. In early implementations of the subroutine feature a store-exit command, usually the first command in the subroutine, deposits the saved address as a jump command at the end of the subroutine. This scheme not only allows for subpictures, but also permit nested subpictures to an indefinite depth.

Now this marks a kind of cardinal point in the wheel of reincarnation. The DEC 340-347 reached this point in design and was still thought to be a display channel. At this level of increasing complexity, however, one should realize and admit that the display data channel is not a mere data channel at all; it is a processor. From here on out one's thinking about the display changes radically.

First of all, one admits that the display's x and y registers form an accumulator and that the display address register is a program counter. What one has is a special purpose computer with a limited and somewhat unusual command repertorire:

Load Immediate and Flash (point)
Add Immediate and Flash (line)
Halt
Jump
Subroutine Jump
Store Subroutine Exit

Taking a broader view, one also realizes that one has a multiprocessor system, with the central processor (the parent computer) and the display processor sharing the same memory. From this viewpoint the Store Subroutine Exit command is a problem since it can change the shared

memory and lead to painful debugging. Another problem is that the subroutine mechanism, useful as it is, does not make it particularly easy to trace one's path back through a multilevel subroutine structure after a light-pen hit.

To solve both these problems, one indulges in a bit more incremental funding and adds a pushdown stack system to the display processor. A subroutine jump stores the return address in the stack and increments the stack pointer. A subroutine return causes a jump to the location stored at the top of the stack and decrements the pointer. All return addresses are stored in one part of memory and one's only concern is to keep the stack from overflowing. Moreover, the contents of the stack give the main processor immediate access in one compact part of memory to the display processor's path through a subroutine hierarchy. As far as we know, the DEC-338 was the first commercial display to include a pushdown stack, and as this is written, the only domestic one[1] with stack hardware.[2]

While all this was going on, one has been adding pushbuttons and keyboards to the display, and has included appropriate registers and flags in the display processor to deal with these, to indicate light-pen hits, to scope edge violations, and the like. All of this information is available to the main processor, but the display processor, which is a rather passive device as we have described it so far, has no way of reacting to button pushes, edge violations, etc. So, for just a little more money, one adds some conditional branch commands that let the display processor test for button pushes, light-pen hits, and so forth. Conditional branch instructions give the display processor the power to do more than merely post complex pictures on the screen. Now it can interact with the user without recourse to the main processor. In fact, with some cleverness, one can write very involved interactive programs for a display processor with conditional branch instructions.

Even with conditionals, the display processor still has a few flaws. For one thing, one would like to make a subroutine transparent to all conditions that may have existed in the calling routine. Transparency is possible for beam position, since subroutines using relative vectors can always return the beam to its initial location, but it is not yet possible for display parameters, such as intensity, character size, and the like, nor for subroutines that use absolute beam positions. So, for a little more money, one makes the stack system a little more elaborate by adding instructions to push the current x and y beam position and the display parameters into the stack, and pop them back.

Now the issue of transparency brings to mind the idea of passing parameters to a subroutine. Parameter passing might be quite useful in display subroutines, and since one can load and store in the pushdown stack, one already has the basic machinery for passing parameters. All that is needed is some way of getting free access to the stack, and all this takes is a means for changing the contents of the stack pointer. So, for very little more money, one adds a command to add to or subtract from the stack pointer.

Thinking about parameters, of course, makes one realize one has been considering local parameters, and it would be nice to have global parameters as well. That is, it would be nice if all parts of a display program could be affected by changing one key word. The convenient way to do this would be to have addressable load and store commands. So, since it won't cost much, why not?

The processor has acquired the following command repertoire:

Load Immediate and Flash	(point)
Add Immediate and Flash	(line)
Halt	
Jump	
Push-Jump	(subroutine)
Conditional Skip	(possibly more than one of these)
Push Parameters	(into stack)
Push X, Y Position	(into stack)
Pop	(restore top item from stack)
Add Immediate to Stack Pointer	
Load	(addressable: C (address) → X, Y)
Store	(addressable: X, Y → C (address))

Many of these commands would be included in a general purpose processor. In fact, to make the display processor general, for just a little more money, one can add:

Execute	(addressable)
Complement	(for subtraction, and logic)
Shift	
Mask	(logical AND, OR, etc.)

And these probably won't add much to the price.

With all these commands, it occurs to one that the display processor could do things like track the light-pen, create "rubber band lines," and handle many other interactive functions that heretofore have been relegated to the main processor. To do these things conveniently, the display processor should have its own interrupt system, and, considering what one has spent so far, that should not cost much to add.

Now where are we? We have built up the display channel until it is itself a general purpose processor with a display. The display is tied directly to its processor; to generate a picture the display processor's central registers are used. In short, we have come exactly once around the wheel of reincarnation.

[1] The British NCR-ELLIOT 4100 is another example.

[2] Graphic II at Bell Telephone Laboratories uses a software approach.

However, we have made some very significant progress during the trip. We have given the processor Load Immediate and Add Immediate commands for displaying points and lines. These operations now take one, rather than three, memory cycles. We have added a pushdown stack system, a mechanism uniquely suited to display subroutining and tracing light-pen hits. In short, we have specially adapted the processor to the task of running a display.

Should we continue around the wheel? We might argue that much of the display processor's power is idle most of the time and that it is wasteful to tie up a general purpose processor merely to refresh a static display. Therefore (for just a little more money) we might consider adding a *channel* to the display processor. We might then consider adding some special commands to the channel to let it follow more complex data structures. If we did so we could move into a second turn around the wheel.

Throughout this discussion we have been assuming that the display processor will operate directly from the memory of the parent computer. The reader should note that we might just as well have started with a display having its own local memory. In either case the wheel of reincarnation works in much the same way. The display processor starts simple and grows until it has become a full computer. Then it gives birth to a second processor which in turn begins to grow.

Looking at some commercial displays, one can find examples at various points around the wheel. As we have said, the DEC Type 30 represents a starting point, while the DEC 340-347 represents about a half-turn. The IDI 10000 series, I.I.I. 1050, Tasker 9000 and the CDC-250 also represent positions less than once around. The IDIIOM represents a full revolution and a quarter, while the DEC 338 represents a revolution and a half. We have found no examples exactly once around the wheel, but we submit this as an interesting design problem: a small general purpose computer with an integrated display system and a single program counter.

3. General Conclusions

It was not until we had traveled around the wheel several times that we realized what was happening. Once we did, we tried to view the whole problem from a broader perspective. We found that some questions had fairly clear answers, but others remained in doubt. The remainder of this paper outlines our conclusions and sets forth the questions we could not answer.

The problem breaks down into two general questions: How closely should the display system be tied to the parent computer? How much computing power should be included in the display processor?

The first question seems simpler to answer than the second. If the display must be located far from the main computer, then the problems of data transmission dictate that it have at least a local memory. Likewise, there are arguments for detaching the display from a parent computer that is running a time-shared system. If the display is too closely coupled to the main machine, competition over memory access and demands from the display for interactive service may degrade the display's or the system's performance. Moreover, if the display processor can change information in memory, there is the danger that it may destroy the time-sharing software.

While a remote display with its own memory seems a good choice for some situations, we feel it has unjustifiable disadvantages unless communication bandwidths force it. We feel a better approach is to locate the display close enough to the main computer so that both can access the same core directly. This approach allows display files to be used in the core where they are prepared; there is no need to ship display data, at a cost of two memory cycles per word, to a remote memory. In interactive situations, this approach makes it easy for the main computer to find out what went on between the display processor, the user, and the display file. Most importantly, particularly in a research system, this approach gives the user the ability to experiment with approaches in which the picture data is merged with other data in his program system. Consequently one of our conclusions has been that the display processor should be closely coupled with the parent computer, that it should take its data from the main computer's core, and that the user should have complete, bit-by-bit control over that data. We recognize that this poses problems in a time-shared system, but we feel the advantages to be gained make it worthwhile to solve them.

If, for geographic or other reasons, one has decided on a tenuous connection between display and main computer, the question of how much power to give the display processor can be answered in terms of how one wishes to use the display. If one plans to display relatively static pictures and can tolerate fairly long delays on interactive services, such as light-pen hits, and button pushes, then there is little point to including general computing power in the display processor. On the other hand, to save memory space, one would probably want to include jump and subroutine commands.

If, by contrast, one wishes to produce more dynamic displays and handle highly interactive situations, then one must at least include general computing power remotely with the display. The question is then whether to integrate the general purpose capability in the display processor itself or to include a separate display channel in the remote device, i.e. whether to go around the wheel of reincarnation exactly once or more than once. Many interactive situations, such as light-pen handling, require that the main display loop be halted, at least while the initial servicing is performed. One could handle these by interrupting the display processor itself. Other functions, such as responding to push buttons, adding to the display file, and interpreting commands from the main computer, can be performed without halting the display. This fact argues

for a display channel combined with a small general purpose computer.

As we have said, we know of no remote display in which the computer and display channel are integrated into one machine, i.e. exactly one turn around the wheel. However, this approach seems to offer some advantages. Having one processor would be cheaper and would eliminate problems arising from the need for communication between two separate processors. By careful interrupt programming the execution time of the slower graphic commands could be utilized for other processing.

Most existing remote displays are based on the second approach, i.e. more than one turn around the wheel. The DEC 338 incorporates a powerful channel with jump, subroutine, and conditional commands in addition to a complete local computer. The Bell Telephone Laboratories Graphic II display[3] represents a different variation of the same approach. Its premise is that in a remote display system, consisting of computer plus display channel, the computer will be idle most of the time and might just as well perform the functions that would otherwise be wired into the channel. The Graphic II channel has a command that interrupts the computer (a PDP-9). The address field of this command indicates what function to perform. Subroutining, conditionals, etc., are done for the display through programs executed by the main computer.

The Graphic II scheme allows great flexibility in building display data structures since the PDP-9 can be programmed to follow almost any structure. However, this flexibility is achieved at a sacrifice in speed. It takes considerably longer to perform jumps, subroutine jumps, etc., by program than by hardware. This time burden could be quite serious, since a single picture may contain many subroutine calls, and all must be repeated each time the picture is refreshed. However, the designer of Graphic II points out that the time burden can be largely eliminated by programs that allow the PDP-9 to follow structure while the display is simultaneously executing graphic commands embedded in the structure.

If it is possible to locate the display processor near to the main computer, we feel, as we have pointed out, that they should share the same memory. In this case, the question of how much display processor to buy becomes rather complicated. No longer is a minimum general purpose capability required. One can choose a design anywhere from a primitive channel to a dedicated general purpose processor plus channel. One way of deciding how much display processor to buy is to look at the jobs the display processor might reasonably be expected to do. There are four.

(1) The display processor must generate pictures from some form of internal representation, which may include multiple calls on display subroutines.

(2) The display processor might generate pictures or picture elements by computation rather than from a static representation in memory. Such pictures as the light-pen tracking cross, point rasters, random points, and arrays of objects are more compactly specified by generation procedures than by listing their elements.

(3) The display processor might provide immediate feedback to the user or handle simple interactive functions such as editing, and light-pen tracking.

(4) The display processor might compile displayable picture representations from higher level data in the user's program system. This would include handling the routine computations required for rotation, scaling, curve generation, and the like, when these are not handled by the display hardware.

As for Job 1, the display processor must certainly follow data structures in core. In our view, a desirable goal is to eliminate the secondary display file that must usually be generated from some higher level structure. The more complex the structures the display processor can follow directly, the more closely, we feel, that goal will be approached. However, in the interest of speed, the display processor must follow structures by executing display commands embedded within the data. It would not be useful, in our view, to give the display processor general computing power merely so that it could *interpret* such structures.

As for Jobs 2 and 3, we feel it does not much matter where the computing power comes from, provided it can be had immediately on demand. One can either provide high level interrupt routines in the main system at risk of degrading the system's performance, or spend the extra money to include the necessary computing power, and possibly an interrupt system in the display processor.

Job 4 does not seem to belong to the display processor at all. As far as generating pictures from data is concerned, we feel the display processor should be a specialized device, capable only of generating pictures from read-only representations in core. A data structure, useful for high level manipulation, represents objects abstractly, and includes, as parameters, the numerical information necessary to generate any particular view. The display processor should be able to follow such structures directly but not generate secondary display files from the information contained in them. Generation of secondary display files is properly the job of the central computer.

The view suggested by Daniel Bobrow that the display processor need not, indeed should not, contain mere general purpose computing power, largely determined the design of our display processor. The design reflects that view most directly in its lack of an addressable store command and in the limitations imposed on access to the stack. For example, information put into the stack can only be returned to the register from whence it came. General computing power, whatever its purpose, should come from the central resources of the system. If these resources should prove inadequate, then it is the system, not the display, that needs more computing power. This decision let us finally escape from the wheel of reincarnation.

RECEIVED AUGUST, 1967; REVISED NOVEMBER, 1967

[3] Ninke, William. Bell Telephone Laboratories, telephone conversation, 11 August 1967.