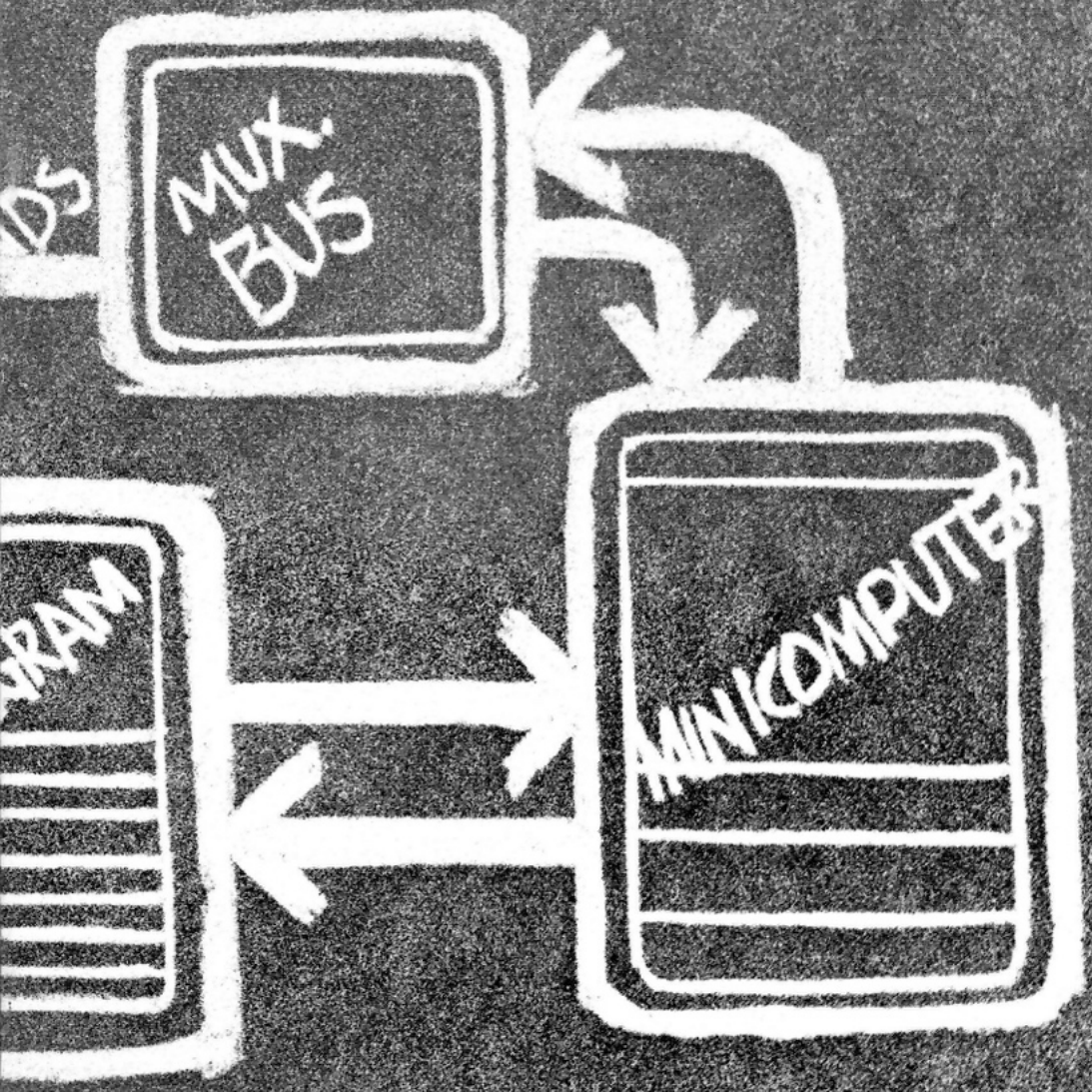
TEKTRONIX 4081
A TECHNICAL REPORT



BY NED THANHOUSER

INTERMIXING REFRESH AND DIRECT VIEW STORAGE GRAPHICS

Ned Thanhouser
Tektronix, Inc.
Beaverton, OR 97005
(503) 638-3411

Presented at SigGraph, 1976

This paper describes an advanced technique in hardware and software for the intermixed display of refresh and storage graphics. Traditional storage tube terminals have the advantage of low cost coupled with the ability to display large amounts of high resolution graphic information, but do not provide for dynamic motion and transformation of a displayed image. High resolution, high density refreshed displays tend to be expensive. Combining both refresh and storage graphics on the same display allows pictures to be segregated into static and dynamic areas to achieve the desired results at reduced cost. The hardware and system software to make this marriage of refresh and storage graphics work effectively for interactive computer graphics is described.

1. INTRODUCTION

The Direct View Storage Tube (DVST) is known for low cost high-resolution, high density graphics. This is achieved by storing images directly on the face plate of the CRT [1], which eliminates the need for image regeneration. Now, by adding a fast constant rate vector generator and a high speed deflaction system, an old feature of DVST's known as "WRITE-THRU" it is now possible to display 1600 centimeters of flicker-free refreshed images along with stored images. [2] Thus, it is now feasible to integrate refresh and storage into a single system, using refresh to display dynamic and temporary images while the static, complex portions of a picture are displayed in storage mode.

Besides integrating storage and refresh capability in hardware, it is also necessary to provide integrated software and system support. The provision of this support is the novel aspect of the system described. The user can construct 'refresh' and 'storage' pictures with the same commands and can use the Graphic Operating System facilities to maintain the structured display list required for refreshed images. [3] This paper describes that support and the techniques used in detail.

2. HARDWARE OVERVIEW

The hardware for mixing refreshed and stored images is diagrammed in figure 1. It consists of a mini-computer, random access memory, a microcoded display controller, and a 19-inch refresh/storage display unit. On command from the mini-computer, the display controller accesses the display list in memory via the high speed Direct Memory Access (DMA) channel. The display list contains beam positioning and status information. The display controller directs the vector generation from which a picture is constructed.

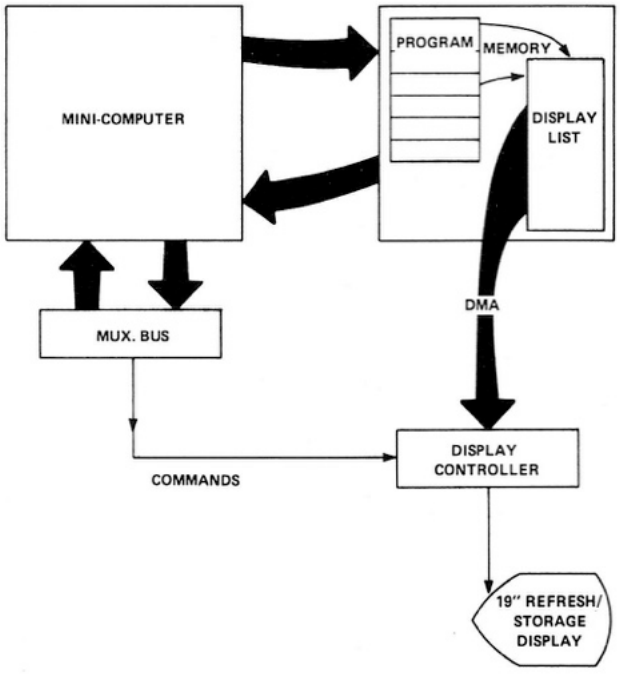


Fig. 1. Hardware for Intermixing Refresh and Direct View Storage Graphics.

3. SOFTWARE OVERVIEW

The software can be broken down into two major areas: Application and System (figure 2). The Application program makes requests to the Graphic Operating system software for graphic output. Before actual display, the graphic output is manipulated by the Graphic Transform Package (GTP). The GTP is responsible for Rotation, Scaling, Clipping, and Window/Viewport Mapping. Once transformed, the

graphic data is passed to the Display Controller Driver (DCDVR), which builds display lists for the display controller.

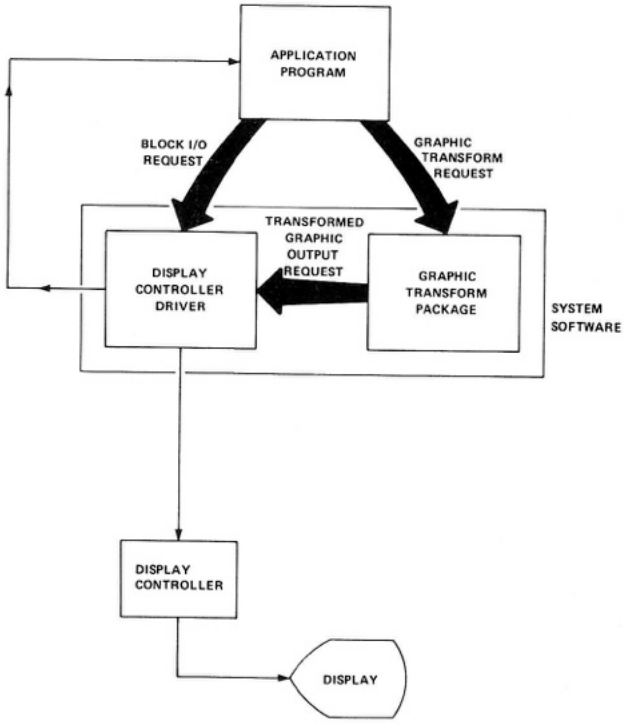


Fig. 2. Software Overview ire Overview

The Display Controller Driver supports a transformed display file [5,6], maintaining separate display code sequences for each individually identified refreshed picture. This allows distinct graphic entities or components to be manipulated independent of one another. Any graphic output request which is not identified with a refreshed object is directed between refresh cycles to the storage display.

The DCDVR (figure 3) consists of three major modules. First is the Display List Generator which is responsible for taking transformed graphic output requests and generating display lists, which are then displayed as stored output or passed to the second module, the Display List Handler. The Display List Handler manages the insertion, deletion, and modification of the refreshed display list. The third module is the Display Controller Interrupt Service Routine, which is responsible for starting up a display list and switching between storage and refresh output. Together, these modules manage the output of storage graphics while maintaining refresh display lists.

4. GRAPHIC OUTPUT

Graphic Output requests to the Graphic Operating system are made with a parameter block as outlined below:

OP-CODE	X	Y

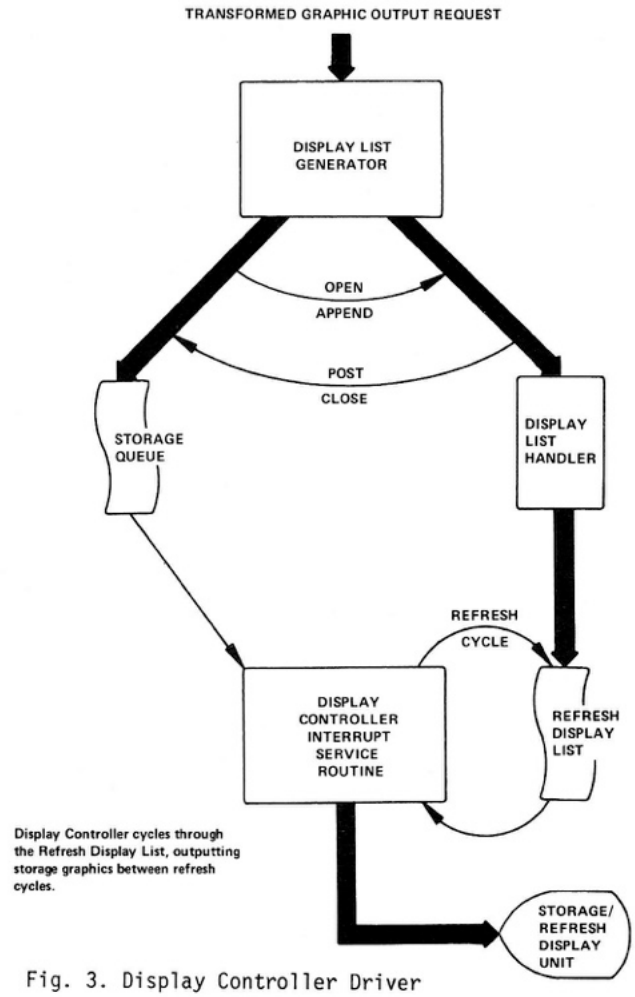


Fig. 3. Display Controller Driver

The OP-CODE specifies the type of graphics (i.e., MOVE, DRAW, DASH, or POINT and ABSOLUTE or RELATIVE), while X and Y fields specify an X-Y coordinate pair to which the beam is to be positioned. For example, to draw a box in storage with sides of 100 units at (0,0) the following sequence of vector requests would be made:

OP-CODE	X	Y
MOVE ABSOLUTE	0	0
DRAW RELATIVE	100	0
DRAW RELATIVE	0	100
DRAW RELATIVE	-100	0
DRAW RELATIVE	0	-100

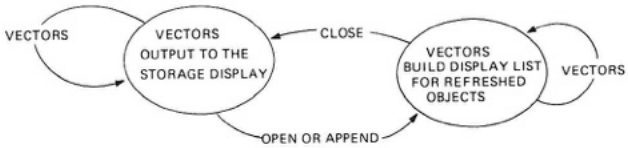
The display controller driver takes these requests and builds display code in fixed length blocks of memory. The interrupt service routine outputs the storage graphics between refresh cycles. When displayed, these blocks of memory are re-used for subsequent storage output requests.

5. REFRESH OBJECT CONSTRUCTION

A sequence of vector output requests, like the one given above, normally generates a stored image. The

display code generated for the picture is traversed once and discarded. However, the user may conditionally save the display code generated by such a sequence as a refreshed object. This is done by preceding vector output requests with an OPEN command. This builds the display code generated by graphic output requests into dynamic memory areas. A CLOSE command stops the display code capturing process, and any further graphic output reverts to the storage display. The just-built 'refresh' object is not visible on the screen but is available for individual manipulation as described under Refresh Object Manipulate.

The process of object construction may be summarized by the following state diagram:



An existing object may have vectors added to it by an APPEND command. A KILL command deletes the object from the system and returns the memory used to hold the display to the dynamic memory pool.

6. REFRESH OBJECT MANIPULATION

Once an object is built, several manipulation commands can be used to affect its viewing status. A POST command links the display list for an object to the refresh display list and thus makes it visible on the screen. An UNPOST removes the object from the display list, making it invisible. A BLINK command causes the object to be automatically POSTed and UNPOSTed at a regular interval. FIX removes the object from the refresh list and then displays it in the storage mode. Figure 4 shows a house partially built by an OPEN sequence, then finished by an APPEND. The house is then make visible with a POST, invisible with an UNPOST, drawn in storage with a FIX, and the display list is discarded with a KILL.

The final object manipulation command, SETPOINT, allows dynamic modification of the initial X and Y coordinates of an object under program control.

The first two coordinates of an object are usually an absolute move, referred to as a setpoint, followed by relative moves and draws. Thus, if the first absolute move's X and Y coordinates are changed, then the rest of the picture will be relocated dynamically. Consider the following example which defines a box as object number 1:

```
OPEN             OBJECT NO. 1
MOVE ABSOLUTE    0,0
DRAW RELATIVE    100, 0
DRAW RELATIVE    0,100
DRAW RELATIVE   -100,0
DRAW RELATIVE    0,-100
POST             OBJECT NO. 1
```

This will draw a box with sides of 100 units in refresh at (0,0) on the display. To move the box (object number 1) to location (500,1000), the following SETPOINT command is given:

```
SETPOINT         OBJECT NO. 1,500,1000
```

The box will immediately appear at the new location.

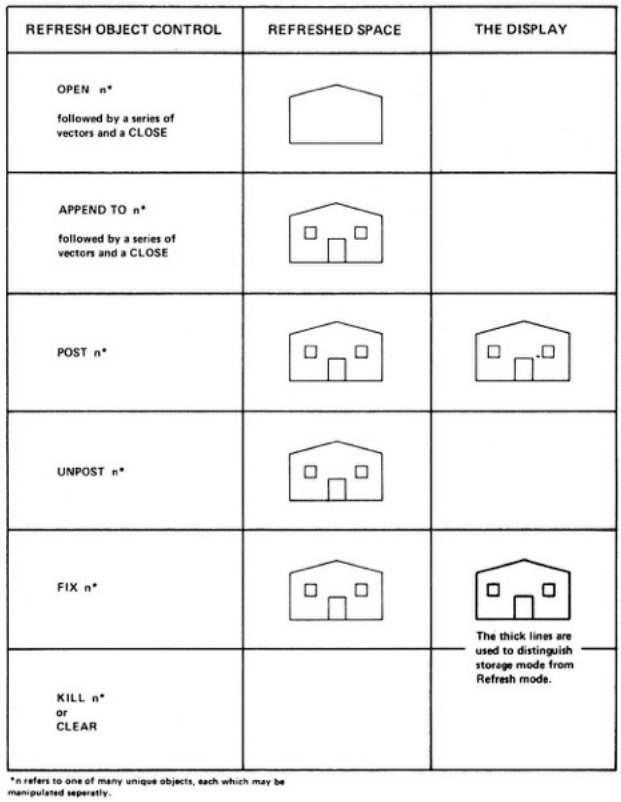


Fig. 4. Refreshed Object Manipulation

Note also what happens if the second vector of an object is a draw absolute, e.g.,

```
OPEN             OBJECT NO. 4
MOVE ABSOLUTE    0,0
DRAW ABSOLUTE    100,100
POST             OBJECT NO. 4
```

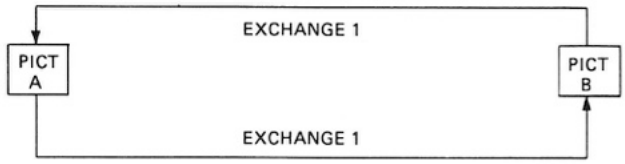
Now, any SETPOINT commands to object number 4 causes the refreshed vector from the first move absolute to the draw absolute to follow like a rubber band, with one end anchored at 100,100. Many variations of this can be used for dragging and rubber band applications.

7. THE DISPLAY CONTROLLER

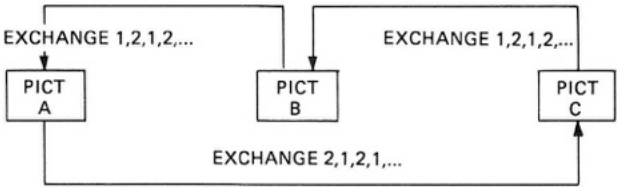
An Exchange Display Status Command initiates the sequence which allows the Display Controller to access the display lists residing in memory.

When an Exchange Status Command is given, the display controller swaps all its current status registers with the contents of the status block in memory. The display code pointed to by the just-loaded block is started. This exchange of status is also used to interrupt a display list to let a new section of display code execute. The first list resumes execution where it was interrupted at the receipt of another Exchange Status Command. Essentially, the Exchange Status may be viewed as a form of co-routine call.

An elementary use of the Exchange Status command would display two different pictures, A and B, each with its own status. One set of display contextual registers would be in the display controller for the current picture, and the other would be in an Exchange Status block. When display of picture A is finished, the current status block is swapped and picture B is started. When picture B is done, another exchange is made and picture A is now swapped in. Note that only one Exchange Status block is needed, since the status of the current picture being displayed is kept in the display controller registers.

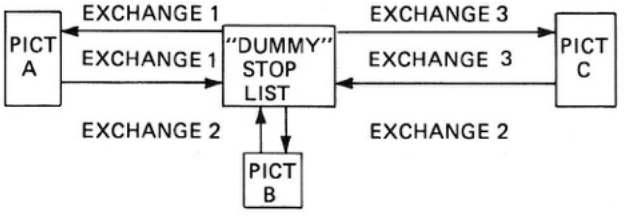


Each box represents an independent segment of display code, while the number following "EXCHANGE" refers to the status block used during the Exchange Status sequence. If more than two segments of display code are to be displayed, more than one status block can be used. For example, with three pictures:



This method is useful if pictures are always swapped on a rotating basis. It has the disadvantage that a different block each time it resumes the same picture. In general, for n pictures, n-1 status blocks would be needed using this scheme.

For a more general use of the Exchange Status command, a "Dummy" display list is used to allow random selection of any segment of display code and to keep each segment associated with one status block. Thus, three pictures could be implemented as follows:



The dummy list contains only a "STOP" and an "INTERRUPT" command. Thus, this use of Exchange Display Status has the following characteristics:

1. Any picture may be interrupted and restarted at any time.
2. Each picture has a fixed status block associated with it.
3. Two Exchange Status Commands must be given to start a different display list. The first references the current picture status block, and the second references the new block. This provides simple, fast selection of logically different pictures.
4. Addition and deletion of pictures is easy in the general case.
5. This method performs the same function as separate 'save status' and 'restore status' commands, but uses only one command.

8. DISPLAY CONTROLLER DRIVER IMPLEMENTATION

The Display controller Driver consists of the three major modules mentioned before: the Display Code Generator (DCG), the Display List Handler (DLH), and the Display Controller Interrupt Service Routine (DCISR). The DCG passes information to the DCISR via the Display Output Queue (DCOQ) (Figure 5). A storage vector output request to the DCG causes display code to be built in fixed length blocks of memory. These blocks are linked together by a display branch command. Once the storage graphics have been displayed, the fixed length blocks are re-used for subsequent storage output requests. Character output causes ASCII to be passed to the DCISR where it is converted to display code and traversed. Commands such as erase or make copy are also passed on the DCOQ (figure 6).

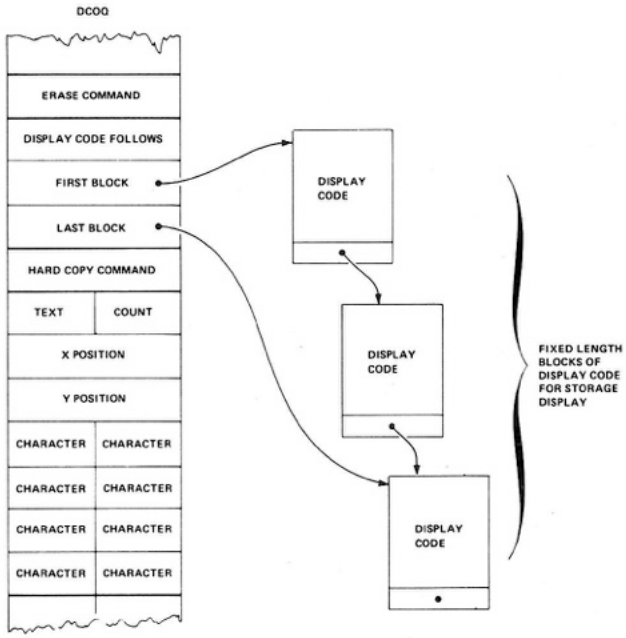


Fig. 5. Display Controller Output Queue Format.

Programs and asynchronous interrupting devices, such as the keyboard, and communications which require single character echoing make entries on the Echo Queue (ECHOQ) by calling the routine ANECHO. In order to prevent input queues from overflowing, this queue is given higher priority than the DCOQ.

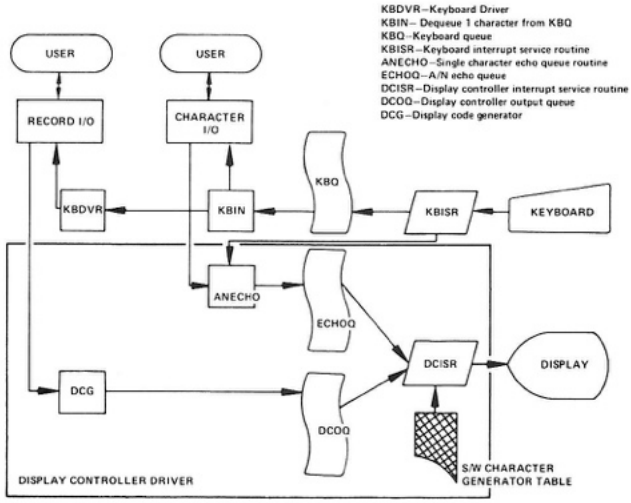


Fig. 6.

Vector output requests to the display code generator which have been preceded by an OPEN or APPEND, cause the display code to be built into fixed length areas of dynamic memory. The Display List Handler (DLH) links these blocks of display code to the refresh list when POSTed, unlinks them when UNPOSTed, etc. A CLOSE or POST command causes all further vector output to be directed to the storage display, until an OPEN or APPEND starts the process again.

Figure 7 shows the data structure used for managing the refreshed objects. Each refreshed object has an entry in the Refreshed Objects Table. Flags are kept in this table for each refreshed object to indicate its status, e.g.,

Existing or Non-existing object
Object under construction, i.e., after OPEN but not CLOSEd
Posted or not posted
Blinking or not blinking
Blink rate and current blink count
Pointer to first and last block of display code

Posting an object is thus: finding the last block of the preceding object, modifying its branch address to the first block of the object to be posted; and setting the branch address in the last block of the object to be posted to the first block of the next posted object. Unposting is a simple reversal of the above. Blinking is accomplished by automatic posting and unposting of an object by the DCISR.

Performing an exchange status for each object would burden the minicomputer with an excessive amount of interrupts if many objects were being displayed. Only one Exchange Status block is used for the refresh list (there are four other Exchange Status blocks: STORAGE OUTPUT, CHARACTER OUTPUT, ECHO OUTPUT, and DUMMY LIST). The first portion of each refreshed object's display code contains a mini-status block. This includes a series of Load Display Controller Status Register Commands to set the correct status for the object. Only one Exchange Status Command to start the display list for all refreshed objects is required. The mini-status block insures that the correct status is set in the display controller for that object. The last object links to an immediate command, stop display list and generate interrupt. This interrupt signals the DCISR that the refreshed display list has been traversed. The DCISR, in this condition, checks for characters to be echoed on ECHOQ, and then for storage output, text or commands on DCOQ. Each of the display requests, e.g., storage output, is started by an Exchange Status Command and ended by a Stop display with Interrupt Command in the display code. If there are no outstanding requests, the "Dummy" list is swapped in.

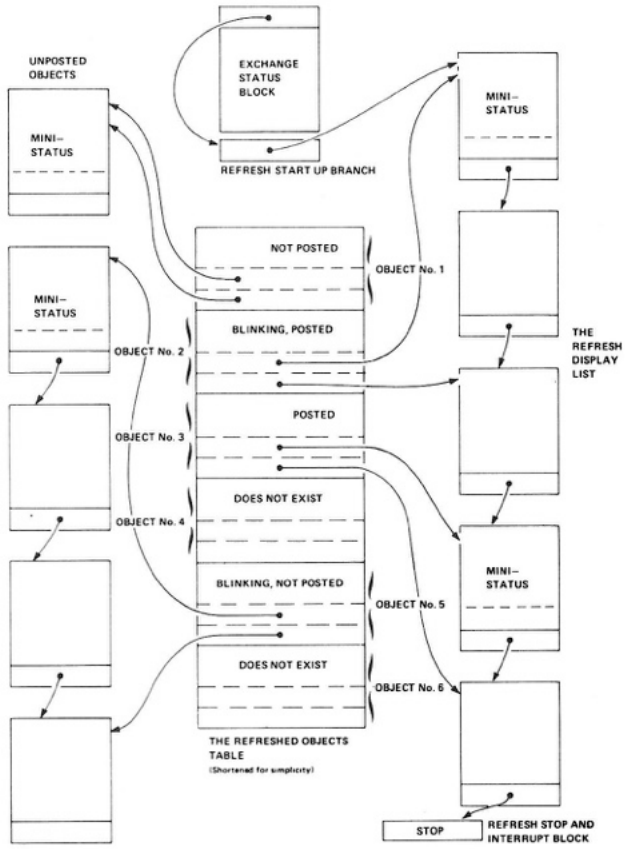


Fig. 7. Refreshed Objects Data Structure

A special refresh timer indicates when the refreshed image should be regenerated to maintain the image on the screen. When it is time for a new refresh cycle, the processor is interrupted and control is given to a special routine in the DCISR. If the refresh list is still running at the time the refresh timer interrupts (i.e., the refresh list takes longer to display than one refresh timer cycle), the interrupt is ignored and flicker is allowed to occur. If no list is running, the DCISR is activated and the refresh list restarted.

Note that all storage output, including character generation, is done between refresh cycles. Our initial design called for interruption of executing

display of a storage list (via an Exchange Status Command) if the refresh timer interrupted. If the refresh list being displayed took longer to display than one refresh cycle, then, at most, a single storage vector would be outputted per refresh cycle. Thus, with a large refresh picture being displayed, only 40 storage vectors per second would be displayed. This technique was abandoned in favor of outputting a large number of storage vectors, uninterrupted, between refresh cycles. Using this technique, intermixing large amounts of refresh and storage does not affect the speed of storage graphic output, nor does it noticeably increase the flicker of the refresh image already exceeding the display cycle time.

9. CONCLUSION

This paper has described the software and hardware used to allow the intermixing of storage graphic output while maintaining a refreshed image. The Exchange Display Status Command provides a great amount of flexibility not discussed here. For example, refresh pictures could be prioritized, and those of a higher priority prevented from flickering by interrupting other lower priority refreshed display lists. Exchange Status might also be used as an alternative method of character echoing. Instead of queuing a character to be displayed at the end of the next refresh cycle, the echoing routine could simply Exchange Status, interrupt the current display list, and start the display list to generate the character echo. When done, another exchange status would resume the original display list. Nonetheless, the implementation described demonstrates that it is possible to intermix Refresh and Direct View Graphics in a consistent and effective manner that is still flexible and easy for an application programmer to use.

ACKNOWLEDGMENTS

Credit must be given to Dean Bailey and Roger Handy for their foresight in the design of the Display Controller hardware and to Larry Koenigsberg who did the initial design and implementation of the Display Controller Driver software. Also, thanks must go to Jon Meads for his invaluable assistance in preparing this paper.

REFERENCES

[1] Anderson, Robert H., Storage Cathode-Ray Tubes and Circuits, Tektronix, Inc., Beaverton, Oregon, 1968.

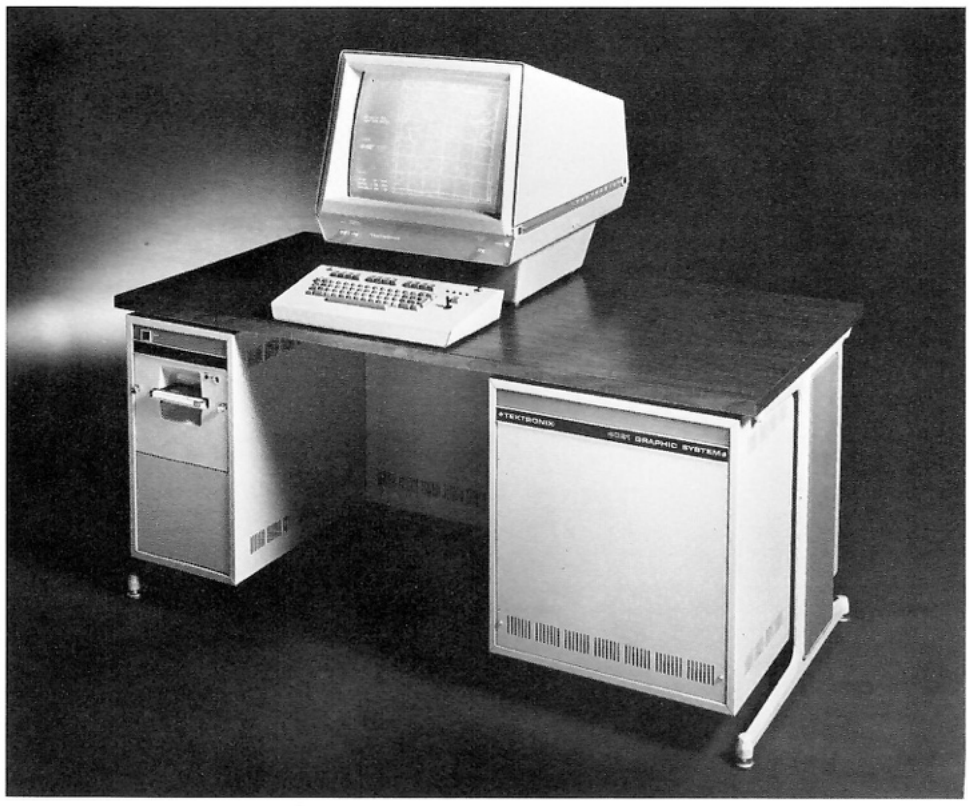
[2] Cheek, Thomas B., "Improving the Performance of DVST Display Systems", 1975 SID International Symposium Digest Technical Papers, April, 1975.

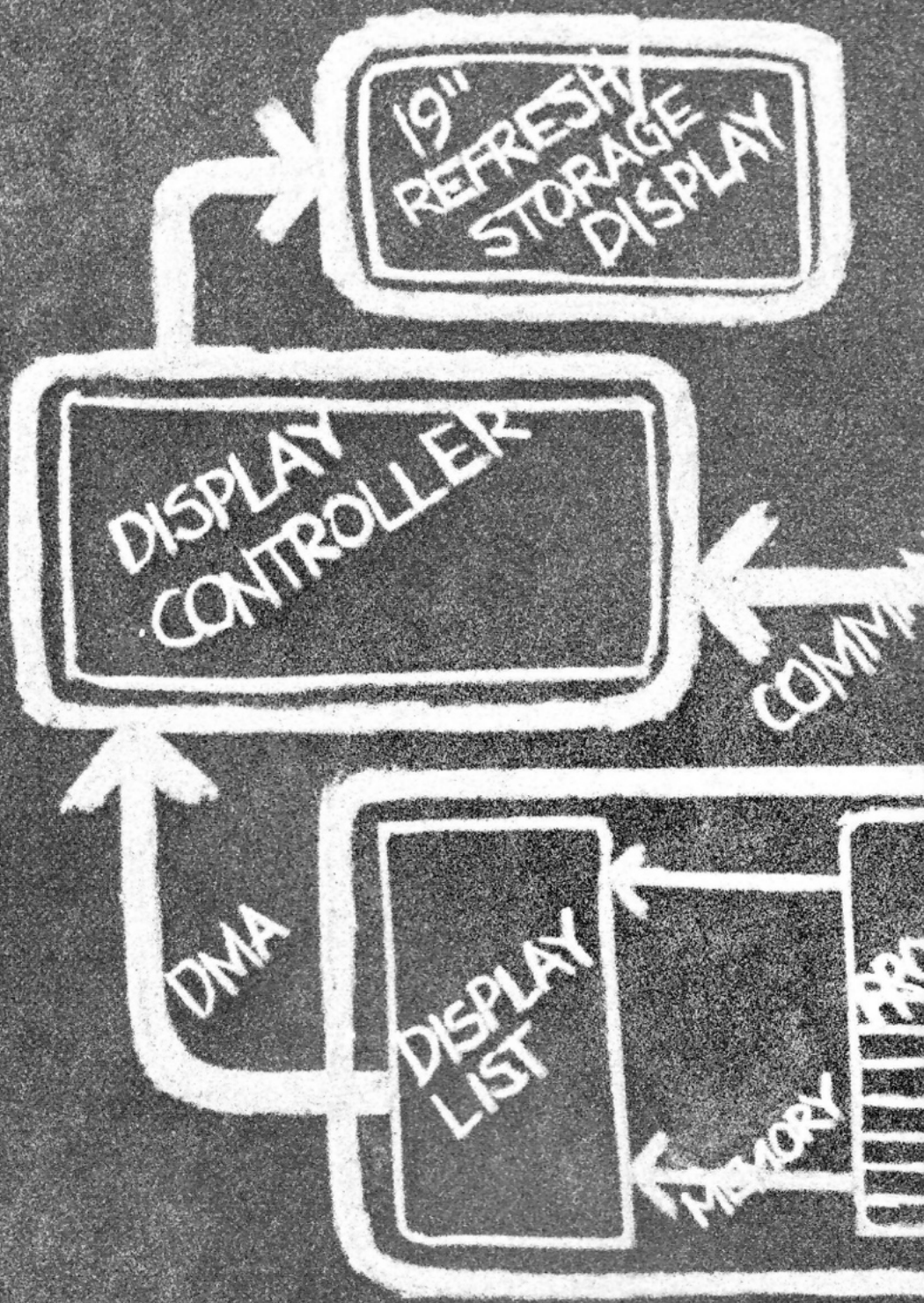
[3] Koenigsberg, L. K., et. al., "A Graphics Operating System", Proceedings of the Second Annual Conference on Computer Graphics and Interactive Techniques, COMPUTER GRAPHICS Siggraph-ACM, Vol. 9, #1, Spring, 1975.

[4] Newman, Wm. and R. F. Sproull, Principles of Interactive Computer Graphics, McGraw-Hill, New York, 1973.

[5] Newman, Wm. and R. F. Sproull, "An Approach to Graphic System Design", Proceedings of the IEEE, Vol 62, #4, April, 1974.

[6] Sproull, R. F. and Elaine Thomas, A Network of Graphic Protocol, Network Graphics Group, ARPA Network Information Center, Stanford Research Institute, 1973.





Tektronix, Inc.
Information Display Group
P.O. Box 500
Beaverton, Oregon 97077
Telephone: (503) 638-3411
Telex: 910-467-8708
Cable: TEKTRONIX

TEKTRONIX®

A-3442