A Virtual Memory System Organization for Bit-Mapped Graphics Displays

Anthony C. Barkans

Schlumberger Technologies, Inc.*
CAD/CAM Division Ann Arbor, Michigan USA

ABSTRACT

Described is a display sub-system, designed for support of a very high speed rendering engine. It provides high-performance graphics to an environment that consists of a hierarchy of resizable windows. The concept of virtual memory has been applied with the organization of the virtual to physical address spaces having a unique mapping that fits the organization of a bit-mapped graphics memory display.

CR Categories and Subject Descriptors: I.3.1 [Computer Graphics]: Hardware Architectures - raster display devices; I.3.3 [Computer Graphics]: Picture/Image Generation - display algorithms.

Introduction

VIEWS, is a new graphics memory system developed to support rendering over one million triangles per second in a full X Windows environment. The limitations of current graphics memory architectures in this environment will first be reviewed. These limitations necessitated a new approach based on the specific graphics memory requirements of high-performance rendering in X. The resulting VIEWS architecture is an innovative and practical solution which exceeded these requirements.

Current Graphics Memory System Design

The designs of hardware memory systems for bit-mapped computer graphics have often followed general-purpose CPU paradigms. The three methods developed for mapping a CPU's address on to its physical memory are direct mapping, segmented virtual memory, and paged virtual memory. All of these methods have more recently been applied to graphics systems as graphics capabilities increase.

Direct mapping of graphics memory to the rendering engine's address space is the textbook example [4] commonly used in current commercial systems. This approach has the lowest cost for a minimum system designed with a frame buffer containing only enough memory to refresh the CRT. Such systems have two performance problems when used in a high performance window environment:

* Currently with: Hewlett Packard, Graphics Technology Division, 3404 Harmony Road, Ft. Collins, Colorado USA

- The rendering engine must clip pixels to the boundaries of windows. These boundaries can be almost arbitrarily complex. As rendering speed increases and more windows are used simultaneously, clipping at full rendering speed becomes very costly.
- As the user moves and deletes windows, exposure events must be sent by the window manager to the graphics applications in other windows. These applications must then redraw at least portions of their windows, using both CPU time and graphics throughput.

Figure 1 shows a single memory plane of such a system and how its contents map to the display surface of the CRT.

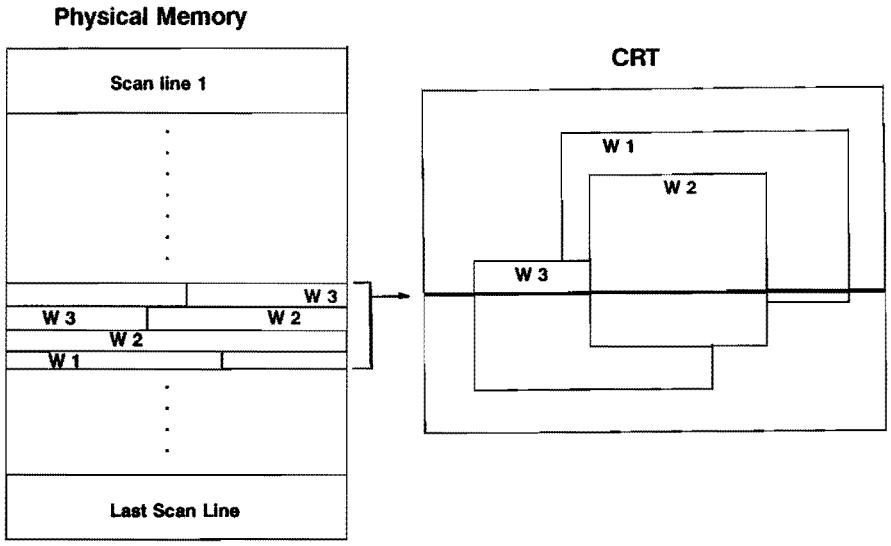

Figure 1: A Direct Mapped Graphics Memory

To address the shortcomings of direct memory mapping in general purpose computer systems, virtual memory systems were introduced. More recently the concept of virtual memory has been applied to graphics memory systems. Several commercially available VLSI chips [2,10] have support for segmented addressing schemes and a complete system using segmented graphics memory has been presented [7].

The use of segmented memory for graphics display sub-systems solves the performance problems inherent in the direct mapping systems. The segments are allocated so that each window has a continuous piece of physical memory of adequate size. Since the entire window is stored in the segment, the rendering engine does not need to clip pixels. The physical memory is large, allowing backing store for windows so that exposure events can be handled without disrupting the application processes and without additional rendering.

However, when segmented graphics memory is used, two new problem areas emerge. The first is the problem encountered in general-purpose CPU segmented memory systems which is also encountered in graphics segmented memory systems. This is complicated memory allocation and large-grained fragmentation. The problem of memory management is compounded as window resize operations are allowed. The second is the assembly problem unique to graphics. Assembling the multi-window display for viewing on the CRT requires accessing physically

disjoint memory locations from different windows to display adjacent pixels. This is not feasible for high resolution displays with complete window capability. An example of a CRT display and the physical memory content for a segmented memory display is shown in Figure 2.

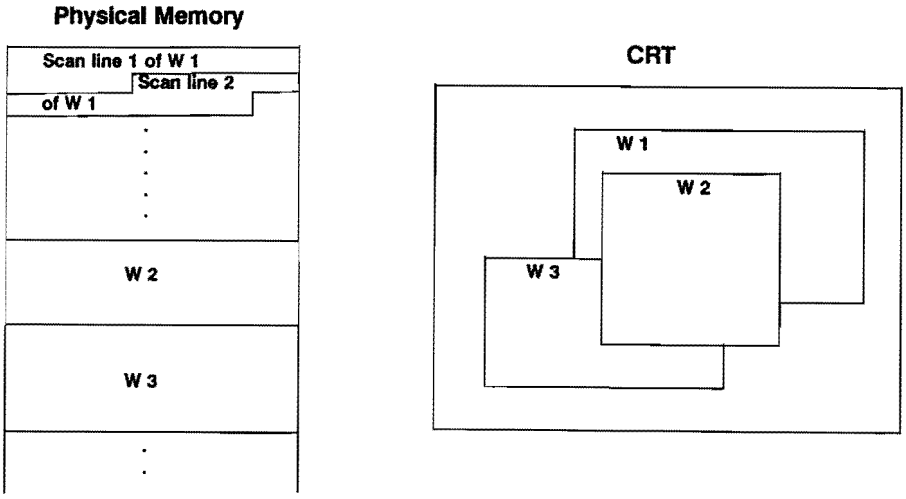

Figure 2: A Segmented Graphics Memory

To expand on the second segmentation problem it should be noted that a random access to physical memory using today's DRAM technology requires about 150 nS, while a high resolution CRT (1280 X 1024 at 60Hz non-interlaced) requires a pixel clock of under 10 nS. If a system allowed an arbitrary number of windows at arbitrary positions, then in the worst case a new window could start at each pixel location and so require a random access to physical memory at the pixel clock speed. Since physical memory can not be accessed at pixel clock speed, systems have been designed where windows must be placed on fixed boundaries [7] or the number of windows that may be displayed at one time are limited [10].

The problems with segmentation were solved in general purpose computer systems by using paged virtual memory systems. A design using a paged virtual memory system for the CPU, where the CPU also serves as the rendering engine, has been presented [1]. In this system graphics rendering used the same datapaths and memory provided for the CPU. Since it was critical to CPU performance, this memory system was designed for both CPU and graphics performance. A second memory was needed for CRT refresh.

Adding paged virtual memory to a dedicated graphics memory system solves the segmentation problems except for the assembly problem. The paging method has another graphics specific problem. The page organization is awkward for graphics. Shown in Figure 3, is a CRT display and the physical memory contents. As can be seen in the figure, Window 1 (W1) has a single scan line that is stored in physically disjoint pages. Note also that the data forming the scan line is not aligned to word boundaries within the page. This makes the control problem more complex when moving graphics data with a small outline (such as a character from a stored font) to an arbitrary location in a large window. This operation is also limited in speed by the inability to fully use the static column access mode of the DRAM chips to increase the transfer rate. The awkwardness of the paged

system is also evident when a window resize operation is performed. As an example, if a window were to be expanded by one pixel on the left side, then the data for the extra pixel on each scan line would have to be inserted at the start of the data for each scan line. This would require reorganizing the data within all pages that form the window. This problem is a consequence of the physical memory organization not mapping to the requirements of windowed graphics.

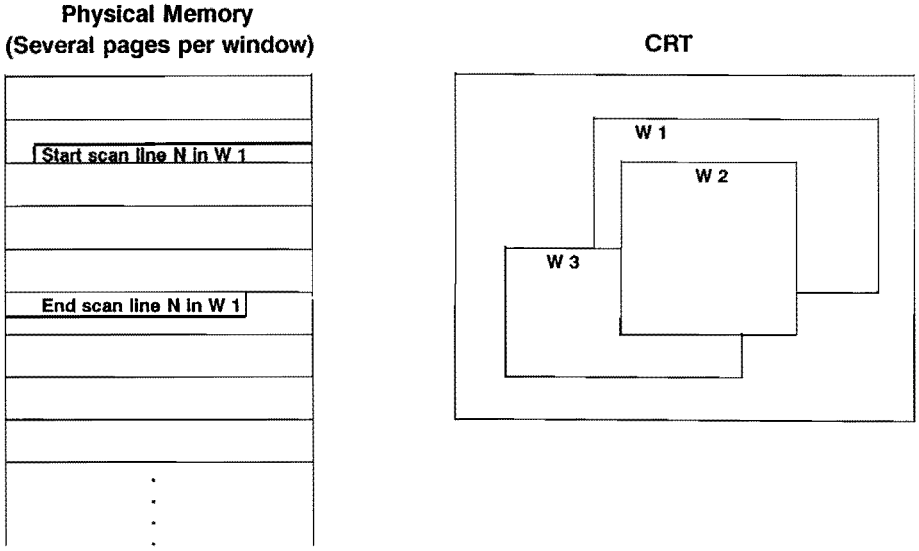

Figure 3: A Paged Graphics Memory

Therefore as graphics memories have developed along the same lines as those of general purpose CPU's, two important problems for graphics systems have not been solved. One is the problem of assembling a multi-window image for CRT refresh and the other is mapping virtual windows onto physical memory in a manner which facilitates high performance graphics operations.

Requirements for High Performance Rendering in the X Windows Environment

Users of modern computer graphics systems are starting to expect the ability to not only run, but to interact simultaneously with multiple applications [8]. At the same time users are requesting more graphics performance. The basic functional capabilities are being provided by building applications to run within window systems such as X Windows [9].

The VIEWS graphics memory sub-system was developed in conjunction with a high speed rendering engine that rendered over one million light-sourced triangles per second [3]. The system could also be used with other high performance rendering hardware [11]. The increased performance that users need is provided by building such advanced rendering hardware.

VIEWS was conceived to support this full rendering speed in an X Windows environment where window manipulations would appear instantaneous to the system user. To meet this design goal without compromise, the development process began with a list of system requirements. The list was broken into three parts: performance requirements, X Windows functional requirements, and advanced graphics feature requirements. This list became the basis for the VIEWS definition.

VIEWS Requirement List

1. PERFORMANCE REQUIREMENTS
   - — In order for the rendering hardware to run at full speed, window clipping had to be hidden from the rendering process.
   - — The memory organization should use the fast access mode (Static Column) of the DRAM chips whenever possible in rendering and data copy operations. This means the structure of the memory chips must be matched to the locality of graphics operations.
   - — Bandwidth requirements for data movement must be minimized. Specifically, pointers should be changed rather than copying data for large data movement operations such as switching buffers in double-buffering.
   - — The organization of data in the memory chips must allow the high speed sequential accesses required for CRT refresh.
   - — The method used to map pixel X,Y addresses to the physical memory array must not be so complicated that memory accesses become a performance bottleneck.

2. X WINDOWS FUNCTIONAL REQUIREMENTS
   - — The display system may not limit the number of windows that can be opened at once.
   - — The display system must allow windows to be placed at arbitrary locations on the display surface.
   - — Windows must be retained with effective utilization of available memory and graceful degradation when memory capacity is exceeded.
   - — Resizing of windows should appear instantaneous to the user.
   - — The backing store should be expandable.

3. ADVANCED GRAPHICS FEATURE REQUIREMENTS
   - — Double buffering must be supported.
   - — Panning within a window must be supported.
   - — Anti-aliasing must be supported by allowing several frames of the same scene to be stored and then combined to produce the final image.
   - — Video output should be flexible, to support several display formats. These would include, a high resolution CRT display, NTSC video, an as yet undefined HDTV format, and a stereo display.
   - — Beyond double buffering the backing store should support many buffers for storage and display of successive frames of animation sequences.

These requirements ruled out using a direct mapped frame buffer, so a virtual memory graphics system was investigated.

The first problems that needed to be solved were the memory organization and assembly problems. These problems are unique to supporting virtual memory in a graphics environment.

To solve the memory organization problems, a new virtual memory mapping scheme was developed. By extending the same mapping technique the assembly problem was solved and ALL the listed requirements for high performance rendering in an X Windows environment were met.

Graphics Virtual Memory to Physical Memory Mapping

The common concept in the three memory organizations already reviewed is the memory space is treated as a linear or 1-dimensional array. This is true in both the physical and virtual spaces. The key to solving the memory mapping problem was to recognize that graphics virtual space is inherently 2-dimensional. VIEWS maps a 2-D virtual address space on to a 1-D physical space. The virtual space is partitioned into rectangular regions, called *tiles*, that map into same-sized regions of the linear physical memory space. The size and shape of these tiles are fixed just as page size in a linear virtual memory system is fixed.

The transformation between the 2-D virtual space and the 1-D physical memory array is done as part of the address translation. In this system a 2-D virtual address is sent to translation hardware where the mapping is done by a combination of a translation RAM and a unique method of hardwiring the address lines through the translation table into the physical memory array. A schematic view of the virtual 2-D address field and its connection to physical memory is shown below as Figure 4. This translation hardware is analogous to the page translation hardware in a CPU with paged virtual memory.

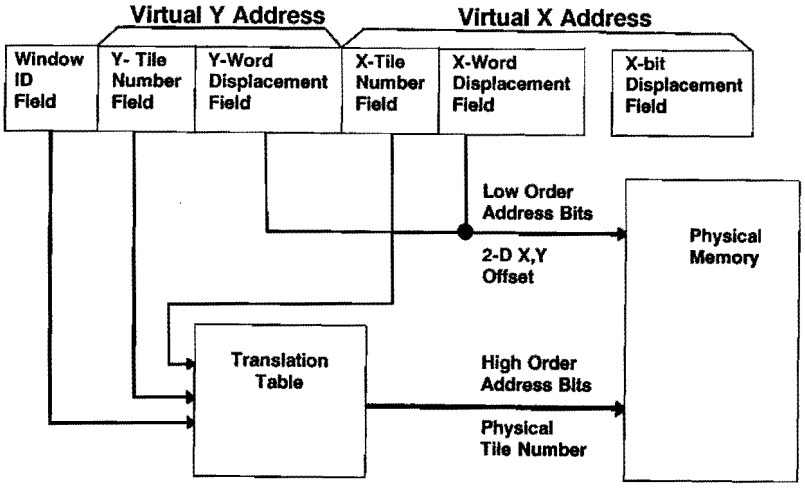

Figure 4: A 2 Dimensional Virtual Address Translated for Physical Memory

From the virtual address field it can be seen that virtual space is treated as a 2-D space, where the X bit displacement, X word displacement, and X tile number form one axis and the virtual Y address forms the other.

In order to go from virtual space to physical space, the virtual X tile number and virtual Y tile number are combined with the window ID to go through the translation table to form part of the physical address. The rest of the physical word address is formed by passing the X word displacement and Y word displacement directly to the memory array. (As is typical in graphics, the X bit displacement is used to build a mask that tells the graphics logic unit which bit to operate on within the word rather than to address a word.)

Note, the field alignment is altered in the translation process. The X word and Y word displacement fields from the virtual address are combined to form the low order field of the physical address. This re-alignment does several things:

- The tile is stored in a linear array of physical memory, but represents a 2-D region of virtual space.
- Data from the same rendering locality (pixels near each other in X and Y) are mapped onto a contiguous sequence of physical memory words. This allows fast static column accesses in the DRAM chips to pixels adjacent in either the X or Y directions.
- The width of a window is removed from the address calculation. In every previous method (direct, segmented, or paged), the data for adjacent scan lines is offset in linear address space by the length of a scan line in that window. In this 2-D mapping, the word offset in physical memory is based simply on the virtual X and Y offset of the pixel from its window origin.

An example of the mapping between the virtual space and the physical memory has been selected such that the 2-D virtual space will map to a Row Address Select (RAS) region in the linear DRAM address space. In other words the X word and Y word displacement fields will be combined to form the Column Address field going to the physical DRAM device. In Figure 5 a single tile from the 2-D graphics space is shown. Shaded in the figure are the pixels that would be stored in the same Row of a single DRAM device. Note that the bit position within the word is obtained from the X bit displacement field.

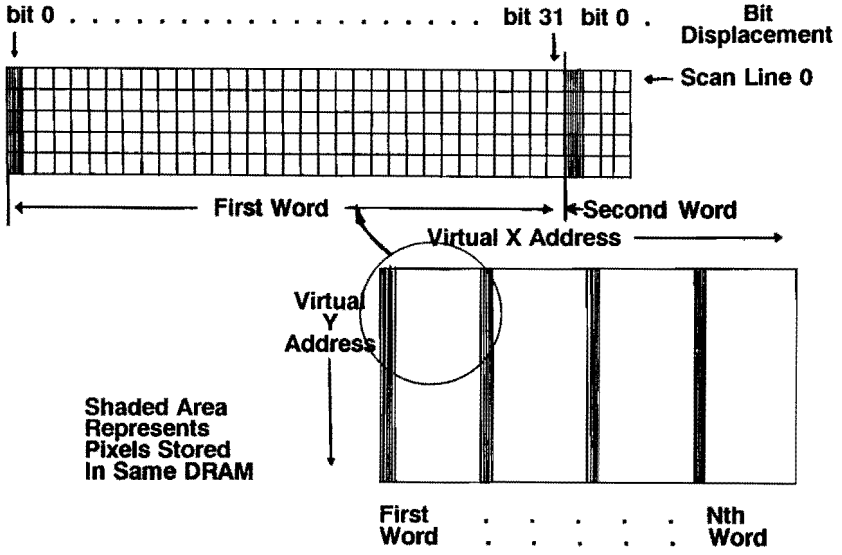

Figure 5: A 2-D Virtual Tile Mapped on to a DRAM array

Figure 5 showed the mapping of the bits within a 2-D tile to a localized region of the address space of the physical devices that form the memory array. Figure 6 expands on Figure 5 by showing how the example tile is held in the DRAM memory array.

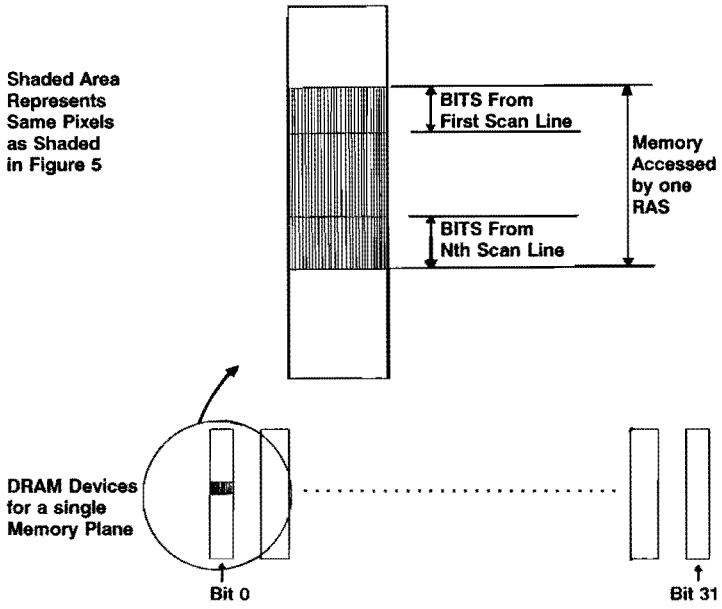

Figure 6: A Tile Stored in the Physical Memory Array

The term *tile* has been used in previous graphics memory system descriptions [5] to denote the array of pixels addressed simultaneously in the DRAM array. The VIEWS system labels this same array of pixels a memory *word*. Data for this group of pixels may be accessed in parallel in a single DRAM access time. In VIEWS this word corresponds to a 1 x 32 horizontal row of pixels in order to match the horizontal scan of both the CRT refresh and the rendering processes.

The VIEWS tile, in contrast, is a new concept analogous to a virtual memory page but with a 2-D organization. In Figure 5, the example 2-D Virtual Tile is the second level of the hierarchy of graphics locality and of DRAM access speed. This large tile will contain the typical small vector, triangle, raster character, or other graphics primitive. It can be accessed by a sequence of fast static column cycles without resorting to full DRAM RAS cycles.

These VIEWS tiles are then organized into virtual 2-D spaces called *canvas buffers* indexed by window ID. A canvas buffer contains all the tiles of a virtual window, whether displayed or not. When a window is opened, a set of physical tiles are allocated for its canvas buffer according to the size of the window. Then the translation table entries are updated to establish the virtual-to-physical mapping.

A schematic representation of a tiled memory array is shown in Figure 7. Window 2 (W2) has been shown filled with a crosshatched pattern on the CRT. The tiles that form the canvas buffer for window 2 are also shown with a crosshatched pattern. Note that the tiles forming the canvas buffer for window 2 may be disjoint in physical memory, just as pages for a linear address space may be disjoint in a conventional CPU with a paged memory system. Also note that within each tile there is a 2-D locality, just as a page has linear locality in the conventional system.

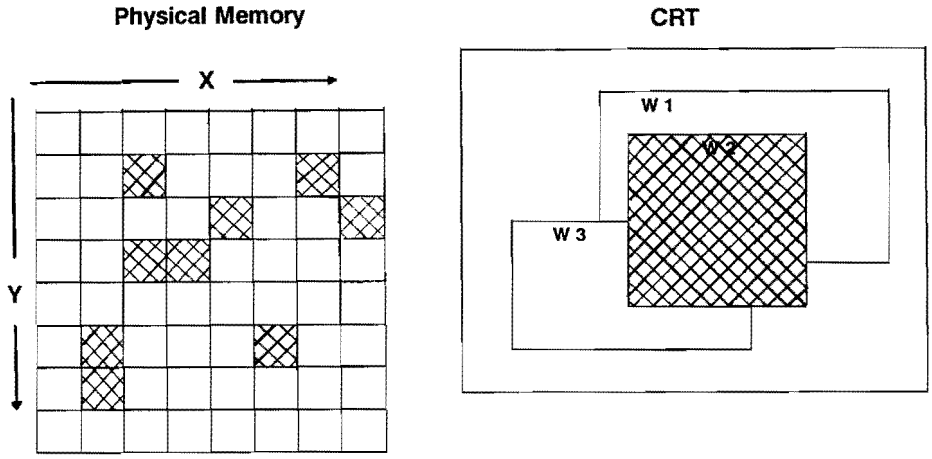

Figure 7: A Tiled Graphics Memory

Extending The Virtual Memory Mapping Concept

The virtual memory organization presented solves the virtual to physical memory mapping problem for graphics display systems. The second problem encountered with adding virtual memory to the graphics system is the data assembly problem.

The assembly problem has been previously addressed. The solutions include replacing the DRAMs used in the frame buffer with fast Static RAMs. Another is to use a separate physical memory to assemble the frame [1]. Also a system that avoids the assembly problem has been presented [12] with the frame store built out of full custom wafer scale memory circuits.

The VIEWS CRT image is located in a contiguous rectangular virtual canvas buffer like any other window. This special canvas buffer, the *scan buffer*, is the same size as the displayed image, with a direct mapping of the virtual pixel X,Y address to the CRT image X,Y location. The scan buffer's image is composed of sections of the various windows visible on the CRT, originating in their own canvas buffers. As any canvas buffer, the physical tiles comprising the scan buffer are scattered in available physical memory.

The traditional frame buffer has a physical mapping to the CRT. In contrast the scan buffer exists in virtual space. However the physical organization of the scan buffer is still critical to CRT refresh.

CRT refresh is accomplished by the sequential accessing of memory words across one row of a single physical tile using static column cycles, followed by a full RAS cycle to get to the next tile. Tile addresses are mapped from adjacent in virtual and CRT image space to disjoint in physical memory space. By requiring the use of DRAM rather than VRAM (dual-port video RAM), this approach necessitates a hardware cost saving. The large memory word and ability to use static column access mode result in adequate DRAM bandwidth for CRT refresh.

The scan buffer must be modified whenever windows are moved or double buffer swaps occur. If the windows happen to be tile-aligned, the scan buffer is modified by simply updating the translation table entries to map different physical tiles to

the CRT image. In this case, the scan buffer requires no additional physical memory to be allocated for CRT refresh.

Windows are not typically tile-aligned. Therefore, tiles of the scan buffer containing non-aligned windows must be allocated an additional physical tile into which the correct data is copied from the appropriate canvas buffer(s). When windows are moved or buffers swapped in double-buffering, the affected scan buffer tiles need to be redrawn by data copying from the original canvas buffers.

When the contents of a window is updated but the window definition does not change, the contents of the updated canvas buffer can be used to update the scan buffer. The tile list that forms the scan buffer is left unchanged. Scan buffer tiles fully covered by an aligned window have already been updated in physical memory. In general, scan buffer tiles are recopied during CRT retrace from the canvas buffer being updated using a high-speed BITBLT capability.

This paper is concerned with the organization of a virtual memory system for bit-mapped graphics. As such, only a brief description of a system designed around the concept will be discussed. First, the functional model will be presented followed by a brief description of the hardware.

VIEWS Functional Model

The window system manager acts as a resource manager. This display sub-system is seen as rendering and display resources. These resources are:

- The Scan Buffer - This serves the function of a traditional frame buffer. The scan buffer is an image of the CRT screen built in a virtual graphics space. The display system micro-processor builds the scan buffer from the visible windows, stored in other tiles, and state information that tells the processor the window priority and screen location. Note that the screen location information could be a list of scan line endpoints for support of irregular shaped windows.
- The Canvas Buffers - These buffers store the entire rendered image for each window. The scan buffer is built by combining parts of several canvas buffers. The system design has two types of canvas buffers. These are the deep (24 bits of color and 4 bits for overlays) and shallow (2 bits deep) canvases. This is consistent with the X Windows scheme for bitmap allocation. The shallow bitmaps are seen as storing alpha-numeric windows and glyphs [fonts].
- Color Look Up Tables were supported.
- The last resource that the window manger has access to in this system is the general purpose micro-coded engine. This processor will process high level commands that the window manger sends. These commands would include window manipulation tasks, and some rendering tasks. The rendering task that would be handled are alpha-numeric updates, some graphics updates and cursor updates.

There will still be cases where the opened windows require more tiles than the system has available. When this happens, tiles can be taken from a window not being updated and used for the current window being rendered. It is important to note that once the visible part of a window is placed in the scan buffer, removing the canvas buffer will not affect the display. Also note, if the scan buffer is then changed and the window that had its canvas buffer removed is now more exposed, then an exposure event will be generated and sent back to the application following the standard X protocol. In this case, the system performance will degrade gracefully, and instantaneous window manipulation will no longer be possible.

VIEWS Hardware Model

Shown in Figure 8 is the high level block diagram of the VIEWS system.

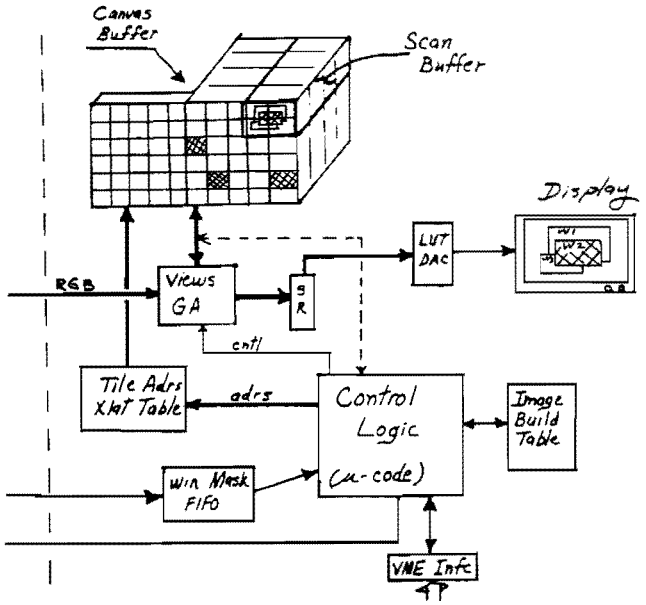

Figure 8: VIEWS Block Diagram

The lines on the left side of Figure 8 show the pixel data and Z status bit entering the VIEWS systems from the supported rendering engine [3].

The VIEWS gate array is an ASIC device that serves as the focal point of data transfers within the VIEWS system. The gate array caches the pixel data [5] from the rendering engine, and is used to store data during high speed data transfers within the memory array (BITBLT). A display FIFO within the VIEWS gate array allows static column mode accesses to be used for reading the tiles that form the scan buffer during CRT updates. The contents of the display FIFO are sequentiality output to the high speed logic section. One VIEWS gate array is used per memory plane. The connection between the gate array and the memory array is 32 bits per plane. This datapath can support a maximum burst transfer rate of 1.12 G bytes per second. There is more than enough bandwidth to allow a full screen (1536 X 1152 pixels) image to be moved within the memory array and the CRT to be updated within one frame time of 16.67 mS.

The Z win mask FIFO caches the Z win bits. These bits are used to condition the control signals going to the VIEWS Gate Array.

The control logic contains all the address generation circuitry, the CRT refresh control circuitry and the micro-coded processor. The micro-processor is responsible for controlling the VIEWS environment, including processing commands from the window manager, initiating and synchronizing VIEWS hardware operations, managing data structures, and performing some low level rendering tasks.

The image build table is the table in the processor's RAM space that defines the way the scan buffer is built.

A simple static ram is used for the tile address translation table. The translation table does not supply status information, such as a valid bit. Instead, the u-processor that controls the sub-system has to keep track of the status of the

memory array. Part of the control process is to make sure that before the rendering engine starts to update a window all the required tiles are present in memory. This requirement assures that tile faults can not occur.

The memory array design supports several physical memory array sizes. The maximum physical memory allowed by the design is 4K X 4K pixels in the deep canvas buffers and scan buffer combined. The shallow buffers can be expanded to 8K X 4K pixels. Note that these limitations are due to the VIEWS implementation and are not architectural limits.

The high speed logic includes ECL shift registers and a LUT/DAC used to drive a high resolution (1536 X 1152 pixels) CRT display.

In VIEWS, the word size is 32 bits, so the bit displacement field is 5 bits long. The tile size is 256 X 256 pixels. The X word displacement field is 3 bits (8 words X 32 pixels/word = 256 pixels). The Y word displacement field is 8 bits. The virtual space is limited to 4096 X 4096 pixels so the X tile displacement and Y tile displacement fields are 4 bits each. The Window ID field is 8 bits long.

The memory array is designed using DRAMs and not VRAMs. The DRAMs cost less and are more flexible on the output port than VRAMs. The control logic allows the refresh to be programmed to any resolution (up to 1536 X 1152 pixels at 60Hz). Some of the programmed resolutions excepted to be used are the very high resolution mode, a high resolution mode (1280 X 1024), and NTSC video. About 1/3 of the high speed BITBLT bandwidth is allocated for the display update task.

Extensions and Limitations to Current System

The system as presented is designed to replace current high end display sub-systems. The architecture would not be cost effective in today's low end graphics systems. A low end implementation of this system would cost about the same as current high end double buffered display sub-systems.

In a low end implementation about half the memory would be needed by the scan buffer, leaving one screen sized piece of physical memory for the deep canvas buffers. An area of several screen sizes, could also be included, for shallow canvas buffers, without adding much cost. Such a system configuration would offer the full functional capabilities of VIEWS, but may not deliver all the performance the architecture can provide, since exposure events would be sent back to the application programs often. One big advantage to using the VIEWS architecture over current designs is that as the next generation of DRAMs become available the memory chips could be changed, while all of the control hardware and software remain untouched. At that time, the physical memory would hold a scan buffer with tiles left over for 3 full screen sized deep canvas buffers.

As new generations of memory chips become commodity items, the price per bit is reduced. The VIEWS memory organization is well suited to take advantage of the evolving DRAM technology. With proper care, a design today would not only plan for the next generation of DRAMs, but allow for many future generations of memory devices.

As larger memory arrays become economically feasible, the simple static ram translation table in VIEWS could be replaced by the more sophisticated translation means used in general purpose virtual memory CPU designs.

Another interesting extension to investigate would be sharing a large memory array between the CPU and graphics memories. This has been done using a linear mapping for both memories [1]. However the extension to this architecture would require a process could request the physical memory map to a either a 2-D or linear virtual space on a process by process basis. In other words both *tiles* and *pages* would be held in the same physical memory array.

Conclusion

A system has been presented, that meets ALL the stated design goals. The system allows high performance graphics in an environment that consists of a hierarchy of resizable windows. By providing instantaneous response to most window operations, the user can feel that he is interacting simultaneously with several applications.

Industry standards and state-of-the-art performance are not mutually exclusive! VIEWS is the first system designed to synergistically combine the industry standard X Window system and state-of-the-art rendering technology for unparalleled interactive graphics.

Acknowledgements

The author would like Jorge Lach to be recognized as co-inventor of the system described. Thanked are Tom Durant for his work on the implementation details, Michael Deering for his review of the system, and Roger Day for both his leadership and moral support. Efforts by the rest of the Applicon engineering crew are greatly appreciated.

The Hewlett Packard Graphics Technology group is thanked for supporting and reviewing this paper, especially Gary Taylor.

REFERENCES

1. **Apgar, Brian et al,** "A Display System for the Stellar Graphics Supercomputer Model GS1000", *Computer Graphics,* **22,** 4, 1988,(Proc SIGGRAPH)

2. **Asal, Mike et al,** "The Texas Instruments 34010 Graphics System Processor", *IEEE CG&A* **6,** 10, (October 1986)

3. **Deering, M. et al,** "The Triangle Processor and Normal Vector Shader: a VLSI System for High Performance Graphics", *Computer Graphics,* **22,** 4, 1988,(Proc SIGGRAPH)

4. **Foley, J. D. and Van Dam, A.,** "Fundamentals of Interactive Computer Graphics", *Addison-Wesley,* 1982

5. **Goris, Andy et al,** "A Configurable Pixel Cache for Fast Image Generation", *IEEE CG&A* **7,** 3

6. **Hamacher, Carl V. et al,** "Computer Organization", *Second Edition McGraw-Hill* 1984

7. **Ilgen, S. and Scherson, I.D.,** "Real Time Virtual Window Management for Bit Mapped Raster Graphics", *Proc. 5th International Conf. on Computer Graphics in Japan, Springer-Verlag, Tokyo* 1987

8. **Lantz, Keith A. et al,** "Reference Models, Window Systems, and Concurrency", *Computer Graphics,* **21,** 2, (April 1987)

9. **Scheifler, Robert W. and Gettys Jim,** "The X Window System", *ACM Trans. Graph.* **5,** 2 (April 1986)

10. **Shires, Glen,** "A New VLSI Graphics Coprocesssor - The Intel 82786", *IEEE CG&A* **6,** 10, (October 1986)

11. **Swanson, Roger and Thayer, Larry,** "A Fast Shaded-Polygon Rnederer", *Computer Graphics,* **20,** 4, 1986,(Proc SIGGRAPH)

12. **Westmore, Richard J.,** "A Window-Based Graphics Frame Store Architecture", *ACM Trans. Graph.* **7,** 4 (October 1987)