FBRAM: A new Form of Memory Optimized for 3D Graphics

Michael F Deering, Stephen A Schlapp, Michael G Lavelle
Sun Microsystems Computer Corporation†

ABSTRACT

FBRAM, a new form of dynamic random access memory that greatly accelerates the rendering of Z-buffered primitives, is presented. Two key concepts make this acceleration possible. The first is to convert the read-modify-write Z-buffer compare and RGBα blend into a single write only operation. The second is to support two levels of rectangularly shaped pixel caches internal to the memory chip. The result is a 10 megabit part that, for 3D graphics, performs read-modify-write cycles ten times faster than conventional 60 ns VRAMs. A four-way interleaved 100 MHz FBRAM frame buffer can Z-buffer up to 400 million pixels per second. Working FBRAM prototypes have been fabricated.

CR Categories and Subject Descriptors: I.3.1 [Computer Graphics]: Hardware Architecture; I.3.3 [Computer Graphics]: Picture/ Image Generation *Display algorithms*; I.3.7 [Computer Graphics]: Three Dimensional Graphics and Realism.

Additional Keywords and Phrases: 3D graphics hardware, rendering, parallel graphics algorithms, dynamic memory, caching.

1 INTRODUCTION

One of the traditional bottlenecks of 3D graphics hardware has been the rate at which pixels can be rendered into a frame buffer. Modern interactive 3D graphics applications require rendering platforms that can support 30 Hz animation of richly detailed 3D scenes. But existing memory technologies cannot deliver the desired rendering performance at desktop price points.

The performance of hidden surface elimination algorithms has been limited by the pixel fill rate of 2D projections of 3D primitives. While a number of exotic architectures have been proposed to improve rendering speed beyond that achievable with conventional DRAM or VRAM, to date all commercially available workstation 3D accelerators have been based on these types of memory chips.

†2550 Garcia Avenue, MTV18-212
Mountain View, CA 94043-1100
michael.deering@Eng.Sun.COM (415) 336-3017
stephen.schlapp@Eng.Sun.COM (415) 336-3818
mike.lavelle@Eng.Sun.COM (415) 336-3103

This paper describes a new form of specialized memory, Frame Buffer RAM (FBRAM). FBRAM increases the speed of Z-buffer operations by an order of magnitude, and at a lower system cost than conventional VRAM. This speedup is achieved through two architectural changes: moving the Z compare and RGBα blend operations inside the memory chip, and using two levels of appropriately shaped and interleaved on-chip pixel caches.

2 PREVIOUS WORK

After the Z-buffer algorithm was invented [3], the first Z-buffered hardware systems were built in the 1970's from conventional DRAM memory chips. Over time, the density of DRAMs increased exponentially, but without corresponding increases in I/O bandwidth. Eventually, video output bandwidth requirements alone exceeded the total DRAM I/O bandwidth.

Introduced in the early 1980's, VRAM [18][20] solved the video output bandwidth problem by adding a separate video port to a DRAM. This allowed graphics frame buffers to continue to benefit from improving bit densities, but did nothing directly to speed rendering operations. More recently, rendering architectures have bumped up against a new memory chip bandwidth limitation: faster rendering engines have surpassed VRAM's input bandwidth. As a result, recent generations of VRAM have been forced to increase the width of their I/O busses just to keep up. For the last five years, the pixel fill (i.e. write) rates of minimum chip count VRAM frame buffers have increased by less than 30%.

Performance gains have mainly been achieved in commercially available systems by brute force. Contemporary mid-range systems have employed 10-way and 20-way interleaved VRAM designs [1][14]. Recent high-end architectures have abandoned VRAM altogether in favor of massively interleaved DRAM: as much as 120-way interleaved DRAM frame buffers [2]. But such approaches do not scale to cost effective machines.

More radical approaches to the problem of pixel fill have been explored by a number of researchers. The most notable of these is the pixel-planes architecture [9][16], others include [7][8][11][4]. [12] and [10] contain a good summary of these architectures. What these architectures have in common is the avoidance of making the rendering of every pixel an explicit event on external pins. In the limit, only the geometry to be rendered need enter the chip(s), and the final pixels for video output exit.

These research architectures excel at extremely fast Z-buffered fill of large areas. They achieve this at the expense of high cost, out-of-order rendering semantics, and various overflow exception cases. Many of these architectures ([16][11][4]) require screen space pre-sorting of primitives before rendering commences. As a consequence, intermediate geometry must be sorted and stored in large batches.

Unfortunately, the benefits from the fast filling of large polygons are rapidly diminishing with today's very finely tessellated objects. That is, the triangles are getting smaller [6]. The number of pixels filled per scene is not going up anywhere near as quickly as the total number of polygons. As 3D hardware rendering systems are finally approaching motion fusion rates (real time), additional improvements in polygon rates are employed to add more fine detail, rather than further increases in frame rates or depth complexity.

3 Z-Buffering and Other Pixel Processing Operations

Fundamental to the Z-buffer hidden surface removal algorithm are the steps of reading the Z-buffer's old Z value for the current pixel being rendered, numerically comparing this value with the new one just generated, and then, as an outcome of this compare operation, either leaving the old Z (and RGB) frame buffer pixel values alone, or replacing the old Z (and RGB) value with the new.

With conventional memory chips, the Z data must traverse the data pins twice: once to read out the old Z value, and then a second time to write the new Z value if it wins the comparison. Additional time must be allowed for the data pins to electrically "turn around" between reading and writing. Thus the read-modify-write Z-buffer transaction implemented using a straightforward read-turn-write-turn operation is four times longer than a pure write transaction. Batching of reads and writes (*n* reads, turn, *n* writes, turn) would reduce the read-modify-write cost to twice that of a pure write transaction for very large *n*, but finely tessellated objects have very small values of *n*, and still suffer a 3-4× penalty.

This is the first problem solved by FBRAM. Starting with a data width of 32 bits per memory chip, FBRAM now makes it possible for the Z comparison to be performed entirely *inside* the memory chip. Only if the internal 32 bit numeric comparison succeeds does the new Z value actually replace the old value. Thus the fundamental read-modify-write operation is converted to a pure write operation at the data pins.

Because more than 32-bits are needed to represent a double buffered RGBZ pixel, some way of transmitting the results of the Z comparison across multiple chips is required. The Z comparison result is communicated on a single external output signal pin of the FBRAM containing the Z planes, instructing FBRAM chips containing other planes of the frame buffer whether or not to write a new value.

The Z-buffer operation is the most important of the general class of read-modify-write operations used in rendering. Other important conditional writes which must be communicated between FBRAMs include window ID compare [1] and stenciling.

Compositing functions, rendering of transparent objects, and anti-aliased lines require a blending operation, which adds a specified fraction of the pixel RGB value just generated to a fraction of the pixel RGB value already in the frame buffer. FBRAM provides four 8-bit 100 MHz multiplier-adders to convert the read-modify-write blending operation into a pure write at the pins. These internal blend operations can proceed in parallel with the Z and window ID compare operations, supported by two 32-bit comparators. One of the comparators supports magnitude tests ($>, \geq, <, \leq, =, \neq$), the other supports match tests ($=, \neq$). Also, traditional boolean bit-operations (for RasterOp) are supported inside the FBRAM. This collection of processing units is referred to as the pixel ALU.

Converting read-modify-write operations into pure write operations at the data pins permits FBRAM to accept data at a 100 MHz rate. To match this rate, the pixel ALU design is heavily pipelined, and can process pixels at the rate of 100 million pixels per second. Thus in a typical four-way interleaved frame buffer design the maximum theoretical Z-buffered pixel fill rate of an FBRAM based system is 400 mega pixels per second. By contrast, comparable frame buffers constructed with VRAM achieve peak rates of 33-66 mega pixels per second [5][14].

Now that pixels are arriving and being processed on-chip at 100 MHz, we next consider the details of storing data.

4 DRAM Fundamentals

Dynamic memory chips achieve their impressive densities (and lower costs) by employing only a single transistor per bit of storage. These storage cells are organized into pages; typically there are several thousand cells per page. Typical DRAM arrays have hundreds or thousands of pages. Per bit sense amplifiers are provided which can access an entire page of the array within 120 ns. These sense amplifiers retain the last data accessed; thus they function as a several thousand bit page buffer. The limited number of external I/O pins can perform either a read or a write on a small subset of the page buffer at a higher rate, typically every 40 ns.

FBRAM starts with these standard DRAM components, and adds a multiported high speed SRAM and pixel ALU. All of this is organized within a caching hierarchy, optimized for graphics access patterns, to address the bandwidth mismatch between the high speed pins and the slow DRAM cells.

5 Pixel Caching

The cache system design goal for FBRAM is to match the 100 MHz read-modify-write rate of the pixel ALU with the 8 MHz rate of the DRAM cells. Figure 1 illustrates this cache design challenge.

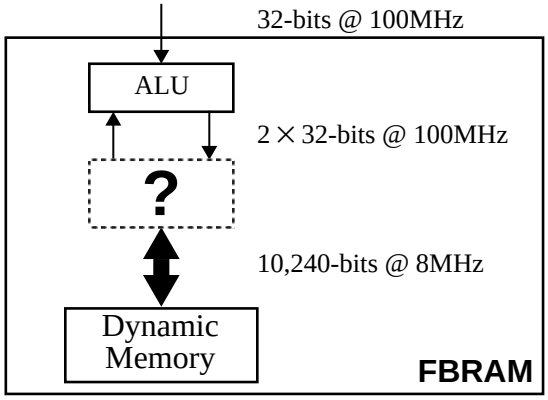

Figure 1. Bandwidth mismatch between pixel ALU and DRAM.

Caches have long been used with general purpose processors; even a small cache can be very effective [17]. But caches have been much less used with graphics rendering systems.

The data reference patterns of general purpose processors exhibit both *temporal* and *spatial* locality of reference. Temporal locality is exhibited when multiple references are made to the same data within a short period of time. Spatial locality is exhibited when multiple references within a small address range are made within a short period of time. Caches also reduce the overall load on the memory bus by grouping several memory accesses into a single, more efficient block access.

Graphics hardware rendering does *not* exhibit much temporal locality, but *does* exhibit spatial locality with a vengeance. Raster rendering algorithms for polygons and vectors are a rich source of spatial locality.

Although the bandwidth available inside a dynamic memory chip is orders of magnitude greater than that available at the pins, this in-

ternal bandwidth is out of reach for architectures in which the pixel cache is external to the memory chips. Others have recognized the potential of applying caching to Z-buffered rendering [13], but they were constrained to building their caches off chip. Such architectures can at best approach the rendering rate constrained by the memory pin bandwidth. As a result, these caching systems offer little or no performance gain over SIMD or MIMD interleaved pixel rendering.

With FBRAM, by contrast, the pixel caches are *internal* to the individual memory chips. Indeed, as will be seen, *two* levels of internal caches are employed to manage the data flow. The miss rates are minimized by using rectangular shaped caches. The miss costs are reduced by using wide and fast internal busses, augmented by an aggressive predictive pre-fetch algorithm.

Each successive stage from the pins to the DRAM cells has slower bus rates, but FBRAM compensates for this with wider busses. Because the bus width increases faster than the bus rate decreases, their product (bus bandwidth) increases, making caching a practical solution.

6 FBRAM Internal Architecture

Modern semiconductor production facilities are optimized for a certain silicon die area and fabrication process for a given generation of technology. FBRAM consists of 10 megabits of DRAM, a video buffer, a small cache, and a graphics processor, all implemented in standard DRAM process technology. The result is a die size similar to a 16 megabit DRAM. A 10 megabit FBRAM is 320×1024×32 in size; four FBRAMs exactly form a standard 1280×1024×32 frame buffer.

Figure 2 is an internal block diagram of a single FBRAM [15]. The DRAM storage is broken up into four banks, referred to as banks A,B,C, and D. Each bank contains 256 pages of 320 words (32 bits per word). Each bank is accessed through a sense amplifier page buffer capable of holding an entire 320 word page (10,240 bits). Banks can be accessed at a read-modify-write cycle time of 120 ns.

Video output pixels can be copied from the page buffer to one of two ping-pong video buffers, and shifted out to the display.

FBRAM has a fast triple-ported SRAM register file. This register file is organized as eight blocks of eight 32-bit words. Capable of cycling at 100 MHz, two of the ports (one read, one write) of the register file allow 10 ns throughput for pipelined 32-bit read-modify-write ALU operations: Z-buffer compare, RGBα blend, or boolean-operations. The third port allows parallel transfer of an entire block (8 words) to or from a page buffer at a 20 ns cycle time via a 256-bit "Global Bus".

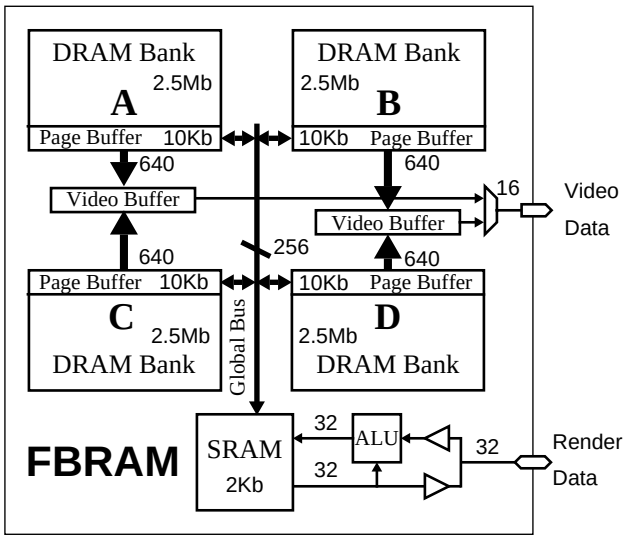

Figure 2. Internal block diagram of a single FBRAM.

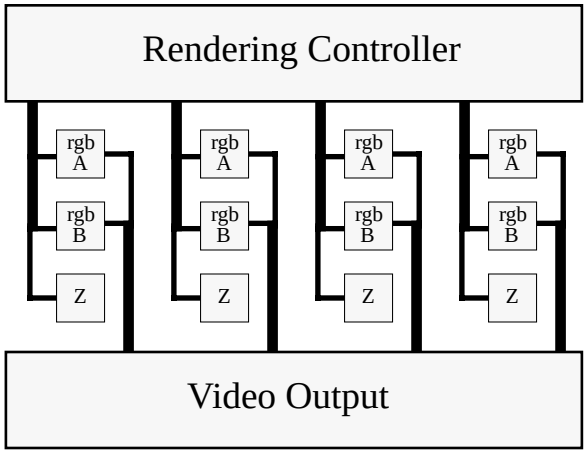

Figure 3. A four-way interleaved frame buffer system composed of 12 FBRAMs (1280×1024, double buffered 32-bit RGBα plus 32-bit Z).

FBRAM has two independent sets of control and address lines: one for the two ALU ports of the SRAM register file; the other for operations involving a DRAM bank. This allows DRAM operations to proceed in parallel with SRAM operations. The cache control logic was intentionally left off-chip, to permit maximum flexibility and also to keep multiple chips in lock step.

7 FBRAM As Cache

Internally, the SRAM register file is a level one pixel cache (L1$), containing eight blocks. Each block is a 2 wide by 4 high rectangle of (32-bit) pixels. The cache set associativity is determined external to the FBRAM, permitting fully associative mapping. The L1$ uses a write back policy; multiple data writes to each L1$ block are accumulated for later transfer to the L2$.

Taken together, the four sense amplifier page buffers constitute a level two pixel cache (L2$). The L2$ is direct mapped; each page buffer is mapped to one of the pages of its corresponding DRAM bank. Each L2$ entry contains one page of 320 32-bit words shaped as a 20 wide by 16 high rectangle of pixels. The L2$ uses a write through policy; data written into a L2$ entry goes immediately into its DRAM bank as well.

The Global Bus connects the L1$ to the L2$. A 2×4 pixel block can be transferred between the L1$ and L2$ in 20 ns.

Four parallel "sense amplifier buses" connect the four L2$ entries to the four DRAM banks. A new 20×16 pixel DRAM page can be read into a given L2$ entry from its DRAM bank as often as every 120 ns. Reads to different L2$ entries can be launched every 40 ns.

8 Four Way Interleaved FBRAM Frame Buffer

The previous sections described a single FBRAM chip. But to fully appreciate FBRAM's organization, it is best viewed in one of its natural environments: a four way horizontally interleaved three chip deep 1280×1024×96-bit double buffered RGB Z frame buffer. Figure 3 shows the chip organization of such a frame buffer, with two support blocks (render controller and video output). Figure 4 is a *logical* block diagram considering all 12 chips as one system. The

discussions of the operations of FBRAM to follow are all based on considering all 12 memory chips as one memory system.

Horizontally interleaving four FBRAMs quadruples the number of data pins; now four RGBZ pixels can be Z-buffered, blended, and written simultaneously. This interleaving also quadruples the size of the caches and busses in the horizontal dimension. Thus the L1$ can now be thought of as eight cache blocks, each 8 pixels wide by 4 pixels high. Taken together, the individual Global Buses in the 12 chips can transfer an 8×4 pixel block between the L1$ and L2$. The four L2$ entries are now 80 pixels wide by 16 pixels high (see Figure 4).

All three levels of this memory hierarchy operate concurrently. When the addressed pixels are present in the L1$, the four way interleaved FBRAMs can process 4 pixels every 10 ns. On occasion, the L1$ will not contain the desired pixels (an "L1$ miss"), incurring a 40 ns penalty ("L1$ miss cost"): 20 ns to fetch the missing block from the L2$ for rendering, 20 ns to write the block back to the L2$ upon completion. Even less often, the L2$ will not contain the block of pixels needed by the L1$ (an "L2$ miss"), incurring a 40-120 ns penalty ("L2$ miss cost") depending upon the scheduling status of the DRAM bank.

This example four way interleaved frame buffer will be assumed for the remainder of this paper.

9 RECTANGULAR CACHES REDUCE MISS RATE

The organization so far shows pixels moving between fast, narrow data paths to slow, wide ones. As can be seen in Figure 4, there is sufficient bandwidth between all stages to, in theory, keep up with the incoming rendered pixels, *so long as the right blocks and pages are flowing*. We endeavor to achieve this through aggressive prefetching of rectangular pixel regions.

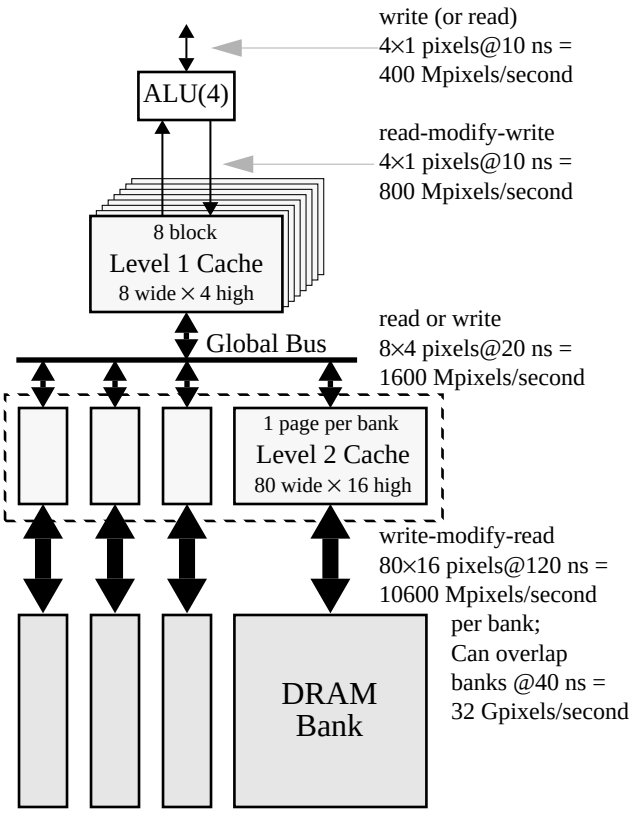

Figure 4. A *logical* representation of a four-way horizontally interleaved frame buffer composed of 12 FBRAMs.

Locality of reference in graphics rendering systems tends to be to neighboring pixels in 2D. Because of this, graphics architects have long desired fast access to square regions [19]. Unfortunately, the standard VRAM page and video shift register dimensions result in efficient access only to long narrow horizontal regions. FBRAM solves this problem by making both caches as square as possible.

Because the L1$ blocks are 8×4 pixels, thin line rendering algorithms tend to pass through four to eight pixels per L1$ block, resulting in a L1$ miss every fourth to eighth pixel (a "miss rate" of 1/4 to 1/8). Parallel area rendering algorithms can aim to utilize all 32 pixels in a block, approaching a miss rate of 1/32.

Similarly, because the L2$ blocks are 80×16 pixels, L2$ miss rates are on the order of 1/16 to 1/80 for thin lines, and asymptotically approach 1/1280 for large areas.

These simplistic miss rate approximations ignore fragmentation effects: lines may end part way through a block or page, polygon edges usually cover only a fraction of a block or page. In addition, fragmentation reduces the effective pin bandwidth, as not all four horizontally interleaved pixels ("quads") can be used every cycle.

FBRAM's block and page dimensions were selected to minimize the effects of fragmentation. Table 1 displays the average number of L1$ blocks (**B**), and L2$ pages (**P**) *touched* when rendering various sizes of thin lines and right isosceles triangles (averaged over all orientations and positions), for a range of alternative cache aspect ratios. The white columns indicate FBRAM's dimensions. Note that smaller primitives consume more blocks and pages *per rendered pixel*, due to fragmentation effects. Although the table implies that a page size of 40×32 is better than 80×16, practical limitations of video output overhead (ignored in this table, and to be discussed in section 13), dictated choosing 80×16.

	Average Pages/Prim				Average Blocks/Prim		
	320×4	160×8	80×16	40×32	32×1	16×2	8×4
10 Pix Vec	2.61	1.84	1.48	1.36	7.57	4.58	3.38
20 Pix Vec	4.21	2.68	1.97	1.71	14.1	8.15	5.76
50 Pix Vec	9.02	5.20	3.42	2.78	33.8	18.9	12.9
100 Pix Vec	17.1	9.42	5.85	4.57	66.6	36.8	24.9
25 Pix Tri	2.96	2.02	1.60	1.46	9.75	6.12	4.68
50 Pix Tri	3.80	2.45	1.89	1.67	13.8	8.72	6.67
100 Pix Tri	4.97	3.05	2.24	1.94	20.0	12.8	9.89
1000 Pix Tri	14.2	8.05	5.41	4.49	82.5	59.6	50.5

Table 1 Average number of Pages or Blocks touched per primitive

10 OPERATING THE FRAME BUFFER

For non-cached rendering architectures, theoretical maximum performance rates can be derived from statistics similar to Table 1. This is pessimistic for cached architectures such as FBRAM. Because of spatial locality, later primitives (neighboring triangles of a strip) will often "re-touch" a block or page before it is evicted from the cache, requiring fewer block and page *transfers*. Additional simulations were performed to obtain the quad, page, and block transfer rates. The left half of Table 2 shows the results for FBRAM's chosen dimensions.

Equation 1 can be used to determine the upper bound on the number of primitives rendered per second using FBRAM. The performance

is set by the slowest of the three data paths (quads at the pins and ALU, blocks on the global bus, pages to DRAM):

$$\text{primitives/sec} = \min\left(\frac{R_Q}{Q}, \frac{R_B}{B}, \frac{R_P}{P}\right) \quad (1)$$

where the denominators **Q**, **B**, and **P** are obtained from the left half of Table 2, and the numerators R_Q, R_B, R_P are the bus rates for quads, blocks and pages. Referring again to Figure 4, R_Q is 100 million quads/sec through the ALU (4 pixels/quad), R_B is 25 million blocks/sec (40 ns per block, one 20 ns prefetch read plus one 20 ns writeback) and R_P is 8.3 million pages/sec (120 ns per page).

		Average Misses/Prim		Million Prim/sec		
	Quad	**B**lock	**P**age	**Q**uad Perf	**B**lock Perf	**P**age Perf
10 Pix Vec	8.75	2.35	0.478	11.4	10.6	17.4
20 Pix Vec	16.4	4.71	0.955	6.10	5.31	8.72
50 Pix Vec	38.9	11.8	2.40	2.57	2.12	3.47
100 Pix Vec	76.7	23.4	4.83	1.30	1.07	1.72
25 Pix Tri	11.6	1.70	0.308	8.62	14.7	27.0
50 Pix Tri	20.2	3.04	0.422	4.95	8.22	19.7
100 Pix Tri	36.1	6.54	0.605	2.77	3.82	13.8
1000 Pix Tri	286.	46.7	4.37	0.350	0.535	1.91

Table 2 FBRAM Performance Limits

The right half of Table 2 gives the three terms of Equation 1. The white columns indicate the performance limit (the minimum of the three for each case).

Equation 1 assumes that whenever the L1$ is about to miss, the rendering controller has already brought the proper block in from the L2$ into the L1$. Similarly, whenever the L2$ is about to miss, the rendering controller has already brought the proper page in from the DRAM bank into the L2$. To achieve such clairvoyance, the controller must know which pages and blocks to prefetch or write back. The FBRAM philosophy assumes that the rendering controller queues up the pixel operations external to the FBRAMs, and snoops this write queue to predict which pages and blocks will be needed soon. These needed pages and blocks are prefetched using the DRAM operation pins, while the SRAM operation pins are used to render pixels into the L1$ at the same time. Cycle accurate simulation of such architectures have shown this technique to be quite effective.

Although pages can only be fetched to one L2$ entry every 120 ns, it is possible to fetch pages to *different* L2$ entries every 40 ns. To reduce the prefetching latency, banks A, B, C and D are interleaved in display space horizontally and vertically, as shown in Figure 5, ensuring that no two pages from the same bank are adjacent horizontally, vertically, or diagonally. This enables pre-fetching any neighboring page while rendering into the current page.

As an example, while pixels of vector **b** in Figure 5 are being rendered into page 0 of bank A, the pre-fetch of page 0 of bank C can be in progress. Usually the pre-fetch from C can be started early enough to avoid idle cycles between the last pixel in page 0 of bank A and the first pixel in page 0 of bank C.

The key idea is that even for vertical vectors, such as vector **d**, we can pre-fetch pages of pixels ahead of the rendering as fast as the rendering can cross a page. Even though vector **c** rapidly crosses three pages, they can still be fetched at a 40ns rate because they are

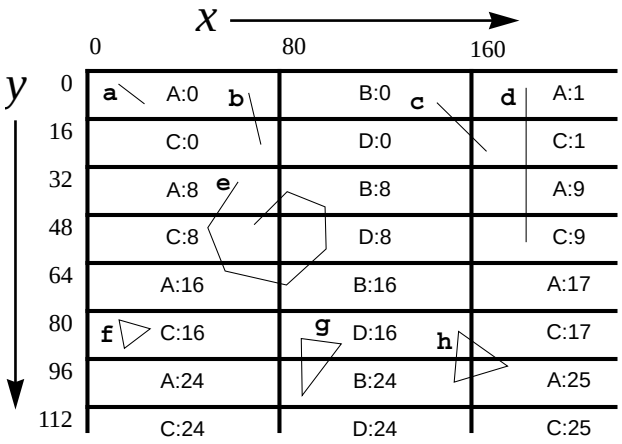

Figure 5. The upper left corner of the frame buffer, showing vertically and horizontally interleaved banks A-D, pages 0-255, and example primitives **a-h**.

from three different banks. Appendix A gives a detailed cycle by cycle example of rendering a 10 pixel vector.

When vectors are chained (vector **e**), the last pixel of one segment and the first pixel of the next segment will almost always be in the same bank and page. Even when segments are isolated, the probability is 75% that the last pixel of one segment and first pixel of next segment will be in different banks, thus enabling overlapping of DRAM bank fetches to L2$.

11 Pixel Rectangle Fill Operations

As fast as the FBRAM pixel write rate is, it is still valuable to provide optimizations for the special case of large rectangle fill. These specifically include clearing to a constant value or to a repeating pattern. Fast clearing of the RGBZ planes is required to achieve high frame rates during double buffered animation.

FBRAM provides two levels of acceleration for rectangle filling of constant data. Both are obtained by bypassing the bandwidth bottlenecks shown in Figure 4.

In the first method, once an 8×4 L1$ block has been initialized to a constant color or pattern, the entire block can be copied repeatedly to different blocks within the L2$ at global bus transfer rates. This feature taps the full bandwidth of the global bus, bypassing the pin/ALU bandwidth bottleneck. Thus regions can be filled at a 4× higher rate (1.6 billion pixels per second, for a four-way interleaved frame buffer).

The second method bypasses both the pin/ALU and the Global Bus bottlenecks, effectively writing 1,280 pixels in one DRAM page cycle. First, the method described in the previous paragraph is used to initialize all four pages of the L2$, then these page buffers are rapidly copied to their DRAM banks at a rate of 40 ns per page. Thus for large areas, clearing to a constant color or screen aligned pattern can proceed at a peak rate of 32 billion pixels per second (0.25 terabytes/sec), assuming a four-way interleaved design.

12 Window System Support

The most important feature of FBRAM for window system support is simply its high bandwidth; however two window system specific optimizations are also included.

Full read-modify-write cycles require two Global Bus transactions: a prefetching read from the L2$, and copyback write to the L2$. Most window system operations do not require read-modify-write cycles when rendering text and simple 2D graphics. For such write-

only operations, the number of Global Bus transactions can be cut in half, improving performance. This is accomplished by skipping the pre-fetching read of a new block from the L2$ to L1$.

Vertical scrolling is another frequent window system operation accelerated by FBRAM. This operation is accelerated by performing the copy internal to the FBRAM. This results in a pixel scroll rate of up to 400 million pixels per second.

13 VIDEO OUTPUT

VRAM solved the display refresh bandwidth problem by adding a second port, but at significant cost in die area. FBRAM also provides a second port for video (see Figure 2), but at a smaller area penalty.

Like VRAM, FBRAM has a pair of ping-pong video buffers, but unlike VRAM, they are much smaller in size: 80 pixels each for a four-way interleaved FBRAM frame buffer vs. 1,280 pixels each for a five-way interleaved VRAM frame buffer. These smaller buffers save silicon and enable a rectangular mapping of pages to the display, but at the price of more frequent video buffer loads.

The FBRAM video buffers are loaded directly from the DRAM bank page buffers (L2$, 80×16 pixels), selecting one of the 16 scan lines in the page buffer. The cost of loading a video buffer in both FBRAM and VRAM is typically 120-200 ns.

To estimate an upper bound for FBRAM video refresh overhead for a 1280×1024 76Hz non-interlaced video display, assume that all rendering operations cease during the 200 ns video buffer load interval. During each frame, a grand total of 3.28 ms (200 ns•1280•1024 pixels / 80 pixels) of video buffer loads are needed for video refresh. Thus 76 Hz video refresh overhead could theoretically take away as much as 25% of rendering performance.

The actual video overhead is only 5-10% for several reasons. First, the pixel ALU can still access its side of the L1$ during video refresh, because video transfers access the L2$. Second, although one of the four banks is affected by video refresh, global bus transfers to the other three banks can still take place. Finally, it is usually possible to schedule video transfers so that they do not conflict with rendering, reducing the buffer load cost from 200 to 120 ns.

For high frame rate displays, the raster pattern of FBRAM video output refresh automatically accomplishes DRAM cell refresh, imposing no additional DRAM refresh tax.

14 FBRAM PEFORMANCE

The model developed in section 10 gave theoretical upper bounds on the performance of a four-way interleaved FBRAM system. But to quantify the performance obtainable by any real system built with FBRAM, a number of other factors must be considered.

First, a 10% derating of the section 10 model should be applied to account for the additional overhead due to video and content refresh described in section 13.

The sophistication of the cache prediction and scheduling algorithm implemented will also affect performance. Equation 1 assumed that the cache controller achieves complete overlap between the three data paths; this is not always possible. More detailed simulations show that aggressive controllers can achieve 75% (before video tax) of the performance results in table 2.

Taking all of these effects into account, simulations of buildable four-way interleaved FBRAM systems show sustained rates of 3.3 million 50 pixel Z-buffered triangles per second, and 7 million 10 pixel non-antialiased Z-buffered vectors per second. FBRAM systems with higher external interleave factor can sustain performances in the tens of millions of small triangles per second range.

All of our simulations assume that the rest of the graphics system can keep up with the FBRAM, delivering four RGBαZ pixels every 10 ns. While this is a formidable challenge, pixel interpolation and vertex floating point processing ASICs are on a rapidly improving performance curve, and should be able to sustain the desired rates.

FBRAM performance can be appreciated by comparing it with the pixel fill rate of the next generation Pixel Planes rasterizing chips [16], although FBRAM does not directly perform the triangle rasterization function. The pixel fill rate for a single FBRAM chip is only a factor of four less than the peak (256 pixel rectangle) fill rate of a single Pixel Planes chip, but has 400 times more storage capacity.

Next let us contrast the read-modify-write performance of FBRAM to a 60 ns VRAM. Assuming no batching, VRAM page mode requires in excess of 125 ns to do what FBRAM does in 10 ns; a 12.5× speed difference.

Batching VRAM reads and writes to minimize bus-turns, as described in section 3, does not help as much as one might think. Typical VRAM configurations have very few scan lines per page, which causes fragmentation of primitives, limiting batch sizes. Table 1 shows that for a 320×4 page shape, a 50 pixel triangle touches 3.8 pages, averaging 13 pixels per page. For a five way interleaved frame buffer, an average of only 2.6 pixels can be batched per chip.

15 OTHER DRAM OFFSHOOTS

A veritable alphabet soup of new forms of DRAM are at various stages of development by several manufactures: CDRAM, DRAM, FBRAM, RAMBUS, SDRAM, SGRAM, SVRAM, VRAM, and WRAM. For 3D graphics, FBRAM is distinguished as the only technology to directly support Z-buffering, alpha blending, and ROPs. Only FBRAM converts read-modify-write operations into pure write operations; this alone accounts for a 3-4× performance advantage at similar clock rates. Other than CDRAM, only FBRAM has two levels of cache, and efficient support of rectangular cache blocks. It is beyond the scope of this paper to derive precise comparative 3D rendering performance for all these RAMs, but FBRAM appears to be several times faster than any of these alternatives.

16 FUTURES

The demand for faster polygon rendering rates shows no sign of abating for some time to come. However, as was observed at the end of section 2, the number of pixels filled per scene is not going up anywhere near as rapidly. Future increases in pixel resolution, frame rate, and/or depth complexity are likely to be modest.

Future predictions of where technology is going are at best approximations, and their use should be limited to understanding trends. With these caveats in mind, Figure 6 explores trends in polygon rendering rate demand vs. memory technologies over the next several years. The figure shows the projected pixel fill rate (including fragmentation effects) demanded as the polygon rate increases over time (from the data in [6]). It also displays the expected delivered pixel fill rates of *minimum chip count* frame buffers implemented using FBRAM and VRAM technologies (extrapolating from Equation 1 and from the systems described in [14][5]). The demand curve is above that achievable inexpensively with conventional VRAM or DRAM, but well within the range of a minimum chip count FBRAM system.

The trend curve for FBRAM has a steeper slope because, unlike VRAM, FBRAM effectively decouples pixel rendering rates from the inherently slower DRAM single transistor access rates. This will allow future versions of FBRAM to follow the more rapidly in-

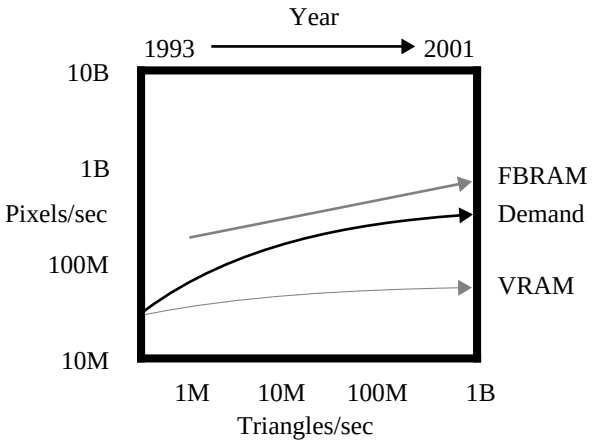

Figure 6. Pixel fill rate needed to match anticipated triangle fill rate demand compared with anticipated delivered pixel fill rate delivered by minimum chip count FBRAM and VRAM systems.

creasing SRAM performance trends. FBRAM still benefits from the inherently lower cost per bit of DRAM technology.

The "excess" pixel fill rate shown for FBRAM in Figure 6 combined with FBRAM's high bit density will permit cost-effective, one pass, full scene antialiasing using super-sampled frame buffers.

17 CONCLUSIONS

In the past, the bandwidth demands of video output led to the creation of VRAM to overcome DRAM's limitations. In recent years, the demands of faster and faster rendering have exceeded VRAM's bandwidth. This led to the creation of FBRAM, a new form of random access memory optimized for Z-buffer based 3D graphics rendering and window system support. A ten fold increase in Z-buffered rendering performance for minimum chip count systems is achieved over conventional VRAM and DRAM. Given statistics on the pixel fill requirements of the next two generations of 3D graphics accelerators, FBRAM may remove the pixel fill bottleneck from 3D accelerator architectures for the rest of this century.

ACKNOWLEDGEMENTS

FBRAM is a joint development between SMCC and Mitsubishi Electric Corporation. The authors would like to acknowledge the efforts of the entire Mitsubishi team, and in particular K. Inoue, H. Nakamura, K. Ishihara, Charles Hart, Julie Lin, and Mark Perry.

On the Sun side, the authors would like to thank Mary Whitton, Scott Nelson, Dave Kehlet, and Ralph Nichols, as well as all the other engineers who reviewed drafts of this paper.

REFERENCES

1. **Akeley, Kurt and T. Jermoluk.** High-Performance Polygon Rendering, Proceedings of SIGGRAPH '88 (Atlanta, GA, Aug 1-5, 1988). In *Computer Graphics* 22, 4 (July 1988), 239-246.
2. **Akeley, Kurt.** Reality Engine Graphics. Proceedings of SIGGRAPH '93 (Anaheim, California, August 1-6, 1993). In *Computer Graphics*, Annual Conference Series, 1993, 109-116.
3. **Catmull, E.** *A Subdivision Algorithm for Computer Display of Curved Surfaces*, Ph.D. Thesis, Report UTEC-CSc-74-133, Computer Science Dept., University of Utah, Salt Lake City, UT, Dec. 1974.
4. **Deering, Michael, S. Winner, B. Schediwy, C. Duffy and N. Hunt.** The Triangle Processor and Normal Vector Shader: A VLSI system for High Performance Graphics. Proceedings of SIGGRAPH '88 (Atlanta, GA, Aug 1-5, 1988). In *Computer Graphics* 22, 4 (July 1988), 21-30.
5. **Deering, Michael, and S. Nelson.** Leo: A System for Cost Effective Shaded 3D Graphics. Proceedings of SIGGRAPH '93 (Anaheim, California, August 1-6, 1993). In *Computer Graphics*, Annual Conference Series, 1993, 101-108.
6. **Deering, Michael.** Data Complexity for Virtual Reality: Where do all the Triangles Go? Proceedings of IEEE VRAIS '93 (Seattle, WA, Sept. 18-22, 1993). 357-363.
7. **Demetrescu, S.** *A VLSI-Based Real-Time Hidden-Surface Elimination Display System*, Master's Thesis, Dept. of Computer Science, California Institute of Technology, Pasadena CA, May 1980.
8. **Demetrescu, S.** High Speed Image Rasterization Using Scan Line Access Memories. Proceedings of 1985 Chapel Hill Conference on VLSI, pages 221-243. Computer Science Press, 1985.
9. **Fuchs, Henry, and J. Poulton.** Pixel Planes: A VLSI-Oriented Design for a Raster Graphics Engine. In *VLSI Design*, 2,3 (3rd quarter 1981), 20-28.
10. **Foley, James, A. van Dam, S. Feiner and J Hughes.** Computer Graphics: Principles and Practice, 2nd ed., Addison-Wesley, 1990.
11. **Gharachorloo, Nader, S. Gupta, E. Hokenek, P. Balasubramanina, B. Bogholtz, C. Mathieu, and C. Zoulas.** Subnanosecond Rendering with Million Transistor Chips. Proceedings of SIGGRAPH '88 (Boston, MA, July 31, Aug 4, 1989). In *Computer Graphics* 22, 4 (Aug. 1988), 41-49.
12. **Gharachorloo, Nader, S. Gupta, R. Sproull, and I. Sutherland.** A Characterization of Ten Rasterization Techniques. Proceedings of SIGGRAPH '89 (Boston, MA, July 31, Aug 4, 1989). In *Computer Graphics* 23, 3 (July 1989), 355-368.
13. **Goris, A., B. Fredrickson, and H. Baeverstad.** A Configurable Pixel Cache for Fast Image Generation. In *IEEE CG&A* 7,3 (March 1987), pages 24-32, 1987.
14. **Harrell, Chandlee, and F. Fouladi.** Graphics Rendering Architecture for a High Performance Desktop Workstation. Proceedings of SIGGRAPH '93 (Anaheim, California, August 1-6, 1993). In *Computer Graphics*, Annual Conference Series, 1993, 93-100.
15. M5M410092 FBRAM Specification. Mitsubishi Electric, 1994.
16. **Molnar, Steven, J. Eyles, J. Poulton.** PixelFlow: High-Speed Rendering Using Image Composition. Proceedings of SIGGRAPH '92 (Chicago, IL, July 26-31, 1992). In *Computer Graphics* 26, 2 (July 1992), 231-240.
17. **Patterson, David, and J. Hennessy.** Computer Architecture: a Quantitative Approach, Morgan Kaufmann Publishers, Inc., 1990.
18. **Pinkham, R., M. Novak, and K. Guttag.** Video RAM Excels at Fast Graphics. In *Electronic Design* 31,17, Aug. 18, 1983, 161-182.
19. **Sproull, Robert, I. Sutherland, and S. Gupta.** The 8 by 8 Display. In *ACM Transactions on Graphics* 2, 1 (Jan 1983), 35-56.
20. **Whitton, Mary.** Memory Design for Raster Graphics Displays. In *IEEE CG&A* 4,3 (March 1984), 48-65, 1984.

APPENDIX A: Rendering a 10 pixel Vector

This appendix demonstrates the detailed steps involved in scheduling FBRAM rendering, using the 10 pixel long, one pixel wide vertical Z-buffered vector shown in Figure 7. This figure shows the

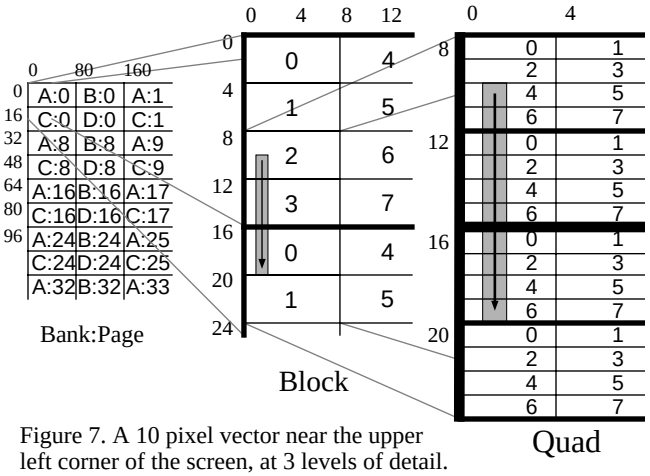

Figure 7. A 10 pixel vector near the upper left corner of the screen, at 3 levels of detail.

memory hierarchy elements touched by the vector at three levels of detail: the coarsest (left most) shows banks (A..D) and pages (0..255), the intermediate detail (middle) shows blocks in the L2$, and the finest (right most) shows pixel quads.

The example vertical vector starts at x=1, y=10, and ends at y=19. Table 3 gives the bank, page, L2$ block, and quad for each pixel in the vector. Note the spatial locality of pixels.

Table 4 below shows the schedule of commands and data issued to the FBRAM, and the resulting internal activities. Note that independent controls are available, and permit parallel L1$ and L2$ activities. The following abbreviations are used in Table 4:

L1$[n]: Block n of the L1$.

L2$[n]: Block n of the L2$.

ACP: Access page (DRAM to L2$ transfer).

X	Y	Bank	Page	L2$ Block	Quad	L2$	L1$
1	10	A	0	2	4	miss	miss
1	11				6	hit	hit
1	12			3	0	hit	miss
1	13				2	hit	hit
1	14				4	hit	hit
1	15				6	hit	hit
1	16	C	0	0	0	miss	miss
1	17				2	hit	hit
1	18				4	hit	hit
1	19				6	hit	hit

Table 3 Bank, Page, L2$ Block, and Quad for each pixel in the vector

RDB: Read block (L2$ → L1$ transfer).

MWB: Masked write block (L1$ → L2$ transfer).

PRE: Precharge bank (free L2$ entry).

read x: Read pixel x from L1$ to ALU.

write x: Write pixel x from ALU to L1$.

We follow the first pixel at (1, 10) through the cache hierarchy. The pixel's page (page 0 of bank A) is transferred to the L2$ entry A in cycles 1 to 4 (notice that the next 5 pixels are transferred too). The pixel's block is then transferred from L2$ entry A to L1$[0] in cycles 5 and 6 (the next pixel is transferred too). The pixel is read from the L1$[0] to the pipelined ALU in cycle 7. The old and new pixels are merged (Z-buffered, blended) during cycles 8 to 11. The resulting pixel is written back to the L1$[0] in cycle 12. The pixel's block is transferred from the L1$[0] back to the L2$ entry A (and DRAM page 0 of bank A) in cycles 14 and 15.

The second pixel at (1,11) hits in both L1$ and L2$, and can follow one cycle behind the first pixel, arriving back in the L1$ in cycle 13. The pixel at (1,12) misses in the L1$, but hits in the L2$, requiring an RDB from L2$ entry A to L1$[1]. The pixel at (1,16) misses in both caches, requiring a L2$ access of bank C, and followed by a transfer from L2$ entry C to L1$[2]. All the other pixels hit in both caches, and are scheduled like the second pixel.

	Commands and Data to FBRAM		Internal Activities												
			L2$ Activities				L1$ Activities								
	L1$ Command and Data	L2$ Command	A	B	C	D	0	1	2	3	4	5	6	7
1		Access Page 0 of Bank A	ACP											
2														
3														
4														
5		L2$[2] of Bank A → L1$[0]	RDB				RDB							
6		Access Page 0 of Bank C			ACP									
7	Merge data with Quad 4 of L1$[0]	L2$[3] of Bank A → L1$[1]	RDB				read 4	RDB						
8	Merge data with Quad 6 of L1$[0]						read 6							
9	Merge data with Quad 0 of L1$[1]							read 0						
10	Merge data with Quad 2 of L1$[1]	L2$[0] of Bank C → L1$[2]			RDB			read 2	RDB					
11	Merge data with Quad 4 of L1$[1]							read 4						
12	Merge data with Quad 6 of L1$[1]						write 4	read 6						
13	Merge data with Quad 0 of L1$[2]						write 6		read 0					
14	Merge data with Quad 2 of L1$[2]	L2$[2] of Bank A ← L1$[0]	MWB				MWB	write 0	read 2					
15	Merge data with Quad 4 of L1$[2]							write 2	read 4					
16	Merge data with Quad 6 of L1$[2]							write 4	read 6					
17								write 6						
18		L2$[3] of Bank A ← L1$[1]	MWB					MWB	write 0					
19									write 2					
20		Precharge Bank A	PRE						write 4					
21									write 6					
22		L2$[0] of Bank C ← L1$[2]			MWB				MWB					
23														

Table 4. Schedule of operations for rendering a 10 pixel vector