BREAKING THE FRAME-BUFFER BOTTLENECK: THE CASE FOR LOGIC-ENHANCED MEMORIES

John Poulton, John Eyles, Steven Molnar, Henry Fuchs
University of North Carolina at Chapel Hill

A central problem in designing interactive graphics systems is the bandwidth bottleneck between the *rasterizer*, the engine that computes pixel values for each primitive in a scene, and the *frame buffer*, the memory system that stores pixel values. Despite remarkable advances in semiconductor storage, this problem not only is still with us, but increasingly is the barrier to further improvements in interactive systems. Current high-end graphics systems are approaching performance limits imposed by the design of commercial DRAMs.

Future graphics applications will require greater geometric complexity and higher quality images. Supporting these applications will require much higher rasterizer performance and much higher bandwidth between rasterizer and pixel memory than can be supported in current designs.

In this paper we first describe the organization of current graphics systems, the performance characteristics of their components, and the demands that future performance requirements will make on these components. We then describe how logic-enhanced memory chips, which tightly couple rasterizing processors to pixel memory, offer a way to improve the back-end performance needed for higher image quality, and show how their design is influenced by the need for object-parallel architectures that support higher rendering rates. We conclude with a concrete example of such a design.

THE PERFORMANCE FRONTIER

Current high-performance graphics systems process a stream of simple geometric primitives (*e.g.*, triangles, lines, and text characters) in a pipeline of processing elements that includes two main functions, as shown in Figure 1.

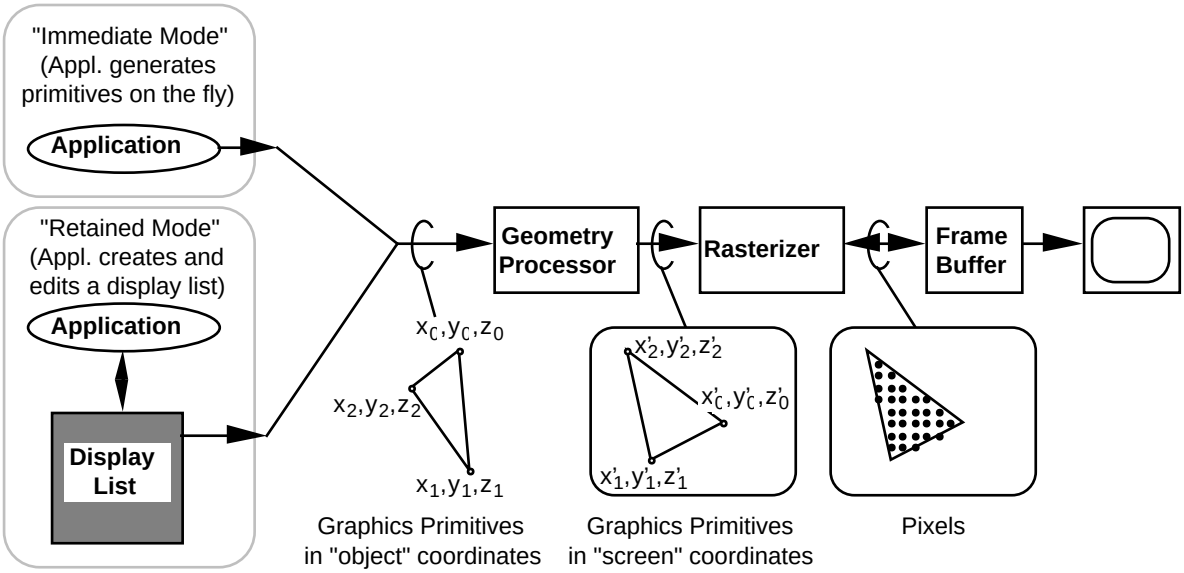


Figure 1: The graphics pipeline.

A *geometry processor* transforms the orientation and perspective of each primitive from an object coordinate system (defined in some arbitrary way convenient for modeling) to screen coordinates, according to the user's viewpoint. These operations transform the geometry information associated with the vertices of a primitive. Screen-space primitive descriptions are then passed to a *rasterizer*, which computes the pixels visible in each primitive and determines the color of each pixel, based on a shading model.

Interactive 3D systems model objects as surfaces tiled by planar primitives, typically triangles, and today's fastest systems can transform and rasterize about 1 million triangles per second with simple Gouraud shading. At these performance levels, bandwidths and computation rates can be estimated as follows:

- **Geometry processing load.** About 200 floating-point operations are needed to transform a triangle from object coordinates to screen coordinates and to perform clipping and lighting. If connected triangles are used, this load is approximately halved. The geometry processor therefore must sustain about 100-200 MFLOPS.
- **Geometry processing bandwidth.** Object-space primitives typically are described in about 10-50 single-precision floating-point words of data, while screen-space primitives require a similar number of words of fixed-point data with sufficient fractional precision to avoid pixel artifacts. Therefore, input and output bandwidths of the geometry processor are roughly equal, about 10-50 MWords/second.
- **Rasterizer load.** Since a typical primitive affects many pixels, there is a data and computation explosion within the rasterizer. The visible primitive at each pixel is generally determined using the *z*-buffer algorithm. Forward differences can be used to calculate *z* values so that only one fixed-point addition is needed per

primitive per pixel. Linearly interpolated (Gouraud) shading can be performed using forward differences as well; one fixed-point addition is needed per color component per pixel. If triangles cover an average of 50 pixels, the rasterizer must perform 200–250 million fixed-point additions per second, in addition to the setup calculations required for the forward differences.

- **Rasterizer/frame buffer bandwidth.** For each pixel in each primitive, a *z* value must be read from the frame buffer. If the pixel is visible, new *z* and color values must be written back to the frame buffer. If we assume the average triangle size is 50 pixels, 3/4 of the pixels are initially visible, and all three color components can be written together, then about 125 million frame buffer accesses per second are required.

Current Graphics System Architectures

Most of today's graphics systems are organized as shown in Figure 2.

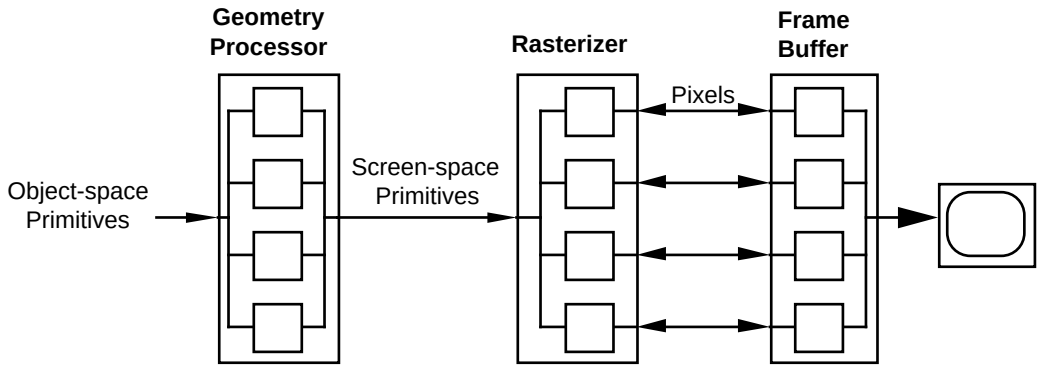


Figure 2: Organization of current graphics hardware systems.

Current floating point microprocessors can achieve only a fraction of the geometry processing speeds, so multiple parallel units are needed. Both MIMD (as in the Hewlett-Packard VRX[1]) and vector or SIMD (as in the Silicon Graphics VGX[2]) are used. Datapath bandwidth in and out of the geometry processor, though large, can readily be handled with single dedicated paths.

Rasterizer performance requirements are so high that parallel solutions are mandatory. Bandwidth into the frame buffer requires multiple paths, and frame buffers, generally built using a combination of DRAM and VRAM memory chips, are organized in parallel banks so that multiple pixels are accessed per memory cycle.

A widely used approach in pixel-parallel rasterizer designs is the *interlaced frame-buffer*[3,4] in which the frame-buffer memory is divided into *m*x*n* partitions, each responsible for every *m*th pixel on every *n*th scan line. A rasterizing processor is

associated with each partition. The number of partitions is smaller than the number of pixels in a typical primitive, so each processor handles multiple pixels per primitive, and on average, the load is well-balanced across the processors, especially if a MIMD organization with buffering is used. Separate memories are sometimes used for *z*-buffer (usually DRAM) and color (usually VRAM); for simple shading, this technique doubles frame-buffer bandwidth.

Current DRAMs and VRAMs can randomly access data and perform the read-modify-write needed for a *z*-buffer at rates of about 5 million operations per second. (Use of page-mode accesses could double this speed, but it is not clear that there is sufficient locality of access to warrant the additional complexity.) Even assuming separate memories for *z* and color, the frame-buffer access rates estimated above require at least 10–20 memory partitions. The Silicon Graphics VGX, one of the fastest general-purpose commercial graphics system, uses 40 partitions to achieve its peak performance of about 1 million flat-shaded triangles per second.

Several recent developments in general-purpose commercial RAMs are of considerable interest to graphics system designers. The proposed Rambus standard[5] could achieve nearly two orders of magnitude higher (peak) access rates, but like VRAMs, access is essentially serial in nature, and is intended primarily for rapidly loading and unloading a cache. Cached DRAMs offer perhaps an order-of-magnitude improvement in access rates, with random access into a small on-chip SRAM. However, since these devices rely on locality of access to achieve their highest speed, they will not greatly accelerate interlaced designs, in which each partition has at most a few pixels in a given primitive. It appears that the newly announced memory devices will improve frame buffer bandwidth only if the graphics pipeline is modified to achieve locality of reference in screen space, as in the screen-subdivision approach described below.

Toward Higher Image Quality

Realistic rendering techniques, such as Phong shading, texturing, and antialiasing are becoming increasingly important features of high-speed graphics systems. Current systems can sustain only a small fraction of their peak rates when executing such algorithms. We summarize the difficulties in implementing each of these algorithms:

- **Phong Shading.** Requires interpolation of surface normals during scan conversion, roughly the same computational load on the rasterizer as interpolating colors for Gouraud shading. Subsequently, however, surface-normal vectors must be renormalized, requiring evaluation of a square root and several multiply/adds per pixel; computing final pixel colors requires two vector dot products and an exponentiation per light source per pixel.
- **Textures.** Texture coordinates must be interpolated during scan-conversion. For perspective-corrected, filtered textures, texture coordinates and texture IDs must be transformed into physical addresses in texture storage (two divides per pixel),

4–8 texture samples must be fetched from texture storage, and the samples must be linearly interpolated, requiring numerous multiplies and adds.

- **Antialiasing.** General methods include variations on the A-buffer algorithm and supersampling. A-buffer algorithms add complexity to the inner-loop of the rasterization algorithm. Supersampling requires rendering the entire scene *k* times, where *k* is the number of samples in the antialiasing filter kernel. Effective antialiasing requires that *k* be at least 5 and preferably 10 or greater, so antialiasing increases the load on the rasterizer by roughly an order of magnitude.

The tremendous rasterizer load imposed by advanced shading calculations can be reduced using *deferred shading*[6,7]. This method avoids shading pixels on surfaces that are ultimately hidden. At scan-conversion time, the minimum information required for shading is calculated and saved in pixel memory. Shading calculations are deferred until rasterization is complete and need only be performed *once per pixel* rather than once per pixel per primitive. The savings are proportional to the scene's average depth complexity.

Based on our group's experimental results, we estimate that rasterizer/frame buffer systems that can generate Phong-shaded, textured images using deferred shading, with antialiasing by supersampling, must compute and access memory at rates 2–3 orders of magnitude higher than today's systems rendering at comparable triangle-per-second rates with simple Gouraud-shading and no antialiasing.

Storage requirements for advanced shading, on the other hand, do not grow as rapidly. Present-day high-performance frame buffers store 50–100 bits per pixel, usually for 1280x1024-pixel screens, and thus need about 8–16 Mbytes of storage. Per-pixel storage needs for advanced shading algorithms will grow modestly—to several hundred or a thousand bits per pixel—while screen resolution may perhaps double.

Future systems that render more primitives per second and support more realistic shading models will be balanced differently from today's systems. Geometry processing will require approximately the same number of calculations per primitive, while rasterization (with antialiasing and advanced shading) will require dramatically more. Deferred shading decouples performance requirements for geometric complexity (number of primitives) from performance requirements for advanced shading. Even with deferred shading, however, computing realistic shading at interactive rates is a formidable task.

Problem Summary

Frame buffers that support interaction with high-quality, antialiased images will have relatively modest storage requirements but extremely high access rates, several orders of magnitude greater than today's systems. The principal difficulty with extending current architectures is the limited bandwidth into commercial memories used to build frame buffers.

Dense dynamic RAM is a commodity product whose design is driven by mass markets in which there is little need to improve the performance parameter of most interest to graphics system designers, *bandwidth per word*. While memory density has been growing exponentially for some years, access rates have improved little better than linearly**,** so bandwidth per word has dropped rapidly with each new generation of DRAM. Recent advances in special high-speed DRAMs rely on locality of data access for improved speed; without significant changes to the design of graphics pipelines, there is not sufficient locality of reference to make effective use of these parts.

How can graphics hardware designers leverage increasing semiconductor memory density to reduce the cost of frame buffers, while at the same time providing the large improvements in bandwidth per pixel needed for advanced image generation techniques? We believe the answer lies in designing memory devices that encapsulate on-chip the high-bandwidth rasterizer/frame buffer interface; these devices should augment, rather than replace, extremely cost-effective commodity DRAMs.

LOGIC-ENHANCED MEMORIES

A straightforward approach to removing the frame-buffer bottleneck is to put the rasterizing processor onto the same chip(s) as the frame buffer memory. This places the bottlenecked interface *inside* the chip, where connections are much cheaper and faster than connections *between* chips.

Much of the advantage of this approach arises from the inherent data parallelism of memory structures. Organized as rectangular arrays of storage cells, memory chips access an entire row of cells during each cycle of operation. But, because chip packages allow only a few I/O pins, only a tiny fraction of this data is available outside the chip. By placing rasterizing processors on the same chips as the memory, they can access entire rows of cell in parallel. Modern RAM cell arrays have several hundred to several thousand cells on a side; so if an entire row is accessed in a single cycle, 2–3 orders-of-magnitude increase in processor-to-memory bandwidth can be obtained, just the leverage we are looking for at the rasterizer/frame buffer interface.

There are additional benefits to this approach. First, custom RAM can operate much faster than commercial memory chips, where much of the access time is spent driving signals on and off the chip and decoding column addresses. Second, the I/O interface of a logic-enhanced memory can be highly optimized for its special purpose. Wide-word access and increased clock speed, taken together, can eliminate the frame-buffer bandwidth bottleneck.

Previous Work

The idea of using custom memory devices for rasterizers and frame buffers is not new. The video RAM (or VRAM)[8] is a simple modification of a DRAM that adds a second

video-data port attached to a long shift register. The shifter can perform loads and stores broadside to an entire row of memory cells. The display is refreshed from the shifter, so refresh can be fully overlapped with conventional memory access, eliminating contention between rasterizer and video subsystems. However, access via this video port is strictly serial, and reads and writes cannot be interleaved.

Pixel-Planes[9] is an extreme solution to the frame-buffer bottleneck, in which the entire frame buffer is built using custom logic-enhanced memories that contain (1) special hardware for parallel evaluation of linear expressions in screen space (equivalent to forward differencing methods used in other systems), (2) tiny pixel-local processors that can perform general arithmetic and logical operations, and (3) all of the pixel memory. This idea was implemented by our group in a full-scale experimental system called Pixel-Planes 4[10]. The system was fast for its time and demonstrated several firsts in interactive rendering: real-time shadows, rapid generation of smooth-shaded spheres, and antialiasing using the technique later called "accumulation buffers". The system was too large and expensive to be practical commercially, however.

A custom memory device for graphics called SLAM (Scan Line Access Memory)[11] processes lines, text, and flat-shaded, stippled polygons. It contains on-chip memory accessed in parallel for a scan-line's worth of pixel data, and its rasterizing processor operates on an entire scan line in parallel. Gouraud shading and *z*-buffer are supported in SLAM by adding a linear-expression tree, identical to that of Pixel-Planes, to interpolate color values. The author built and demonstrated a small prototype SLAM system, but it was not commercialized.

A custom memory device called SAGE (Systolic Array Graphics Engine)[12] passes descriptions of spans (intersections of primitives with a scan line) from left to right across an array of processors associated with the pixels of a scan line. Every operating cycle, each pixel processor receives a new span description, updates its pixel data based on the span description, and passes the span description to the right. SAGE's main component, a large custom memory-intensive chip, was fabricated and tested, but the idea was not developed as a product.

Our group completed a second experimental system, Pixel-Planes 5, in 1990[13]. Using logic-enhanced memories much like those of our earlier system, but with a different system architecture consisting of 128x128-pixel "region processors", it solved two of the problems identified in Pixel-Planes 4: how to use logic-enhanced memories more efficiently than instantiating the entire screen, and how to process objects in parallel on multiple rasterizers.

Although several experimental graphics systems based on logic-enhanced memories have been built, no commercial systems have used this approach to date. Instead, nearly all high-end commercial graphics systems use interleaved frame-buffers with custom rasterizer ASICs (application-specific integrated circuits) controlling each of several banks of standard DRAM/VRAM memories. Rasterizer ASICs have been built using a variety of design and fabrication styles, from gate array and standard cell semi-custom to

full custom hand layout. This general approach has been cost effective, but as we have discussed above, it may be difficult to extend to much higher performance levels.

A frequent argument against using logic-enhanced memories in rasterizers is economic; the cost per bit for custom memories is far higher than for commodity DRAMs, so frame buffers built from such memories cannot compete with conventional frame buffers. However, as we will discuss in the next section, it is possible to obtain most of the benefits of fast on-chip custom memory with a *pixel buffer* holding only a fraction of the screen pixels. It is widely perceived that ASICs are the only economical way to implement the demanding special functions needed in a rasterizer. The cost of this custom silicon depends mainly on chip area and design time; we argue that including modest amounts of fast local memory on rasterizer ASICs can increase their performance dramatically, without comparable increases in cost.

ARCHITECTURAL ISSUES

There are two critical issues in the design of logic-enhanced memories for graphics: First, memory density in ASICs is limited, generally several generations behind commercial DRAMs, so custom memory must be used as efficiently as possible. Second, the rasterizing processors must be organized to exploit the data parallelism provided by on-chip memory.

Memory Organization

ASIC fabrication processes are intended to implement mainly logic functions and lack the special structures needed to build dense storage cells. DRAM processes are tuned for volume, not diversity, and are therefore unavailable to custom memory designers. Memory on ASICs therefore is less dense and consumes far more energy per bit than commercial DRAMs. To live within current economic constraints, custom memory-intensive rasterizer ASICs can have at most perhaps a few hundred kilobits of fast local storage on a few dozen chips and require a separate frame buffer built with commercial DRAM/VRAM.

Caching is a standard technique for coupling small, fast memory close to a processor with larger, slower bulk memory. Effective caching requires data locality, however. Cache schemes for graphics, which depend on data locality within the pixels of a single primitive, have been proposed, and some have been built[14,15]. In conventional graphics pipelines, where successive primitives may be far apart in screen coordinates, further locality of reference is not available.

Locality can be guaranteed by sorting primitives into fixed-size screen regions, and having a single region-sized *pixel buffer* within the rasterizer. If all primitives that fall into a given region are processed together before proceeding to the next region, each region need be processed only once. After the region is completely rasterized,

intermediate values can safely be discarded, so data need only flow *out* of the pixel buffer, and the data need only include final pixel colors. If the rasterizer is guaranteed to be fast enough, no frame buffer is needed. If a frame buffer is used, it only has to be large enough to store colors. SLAM and SAGE, like other scan-line oriented designs, use one variation of this approach: primitives are sorted by *y*-coordinates according to the scan lines they cover, and custom memory is provided for a single scan line.

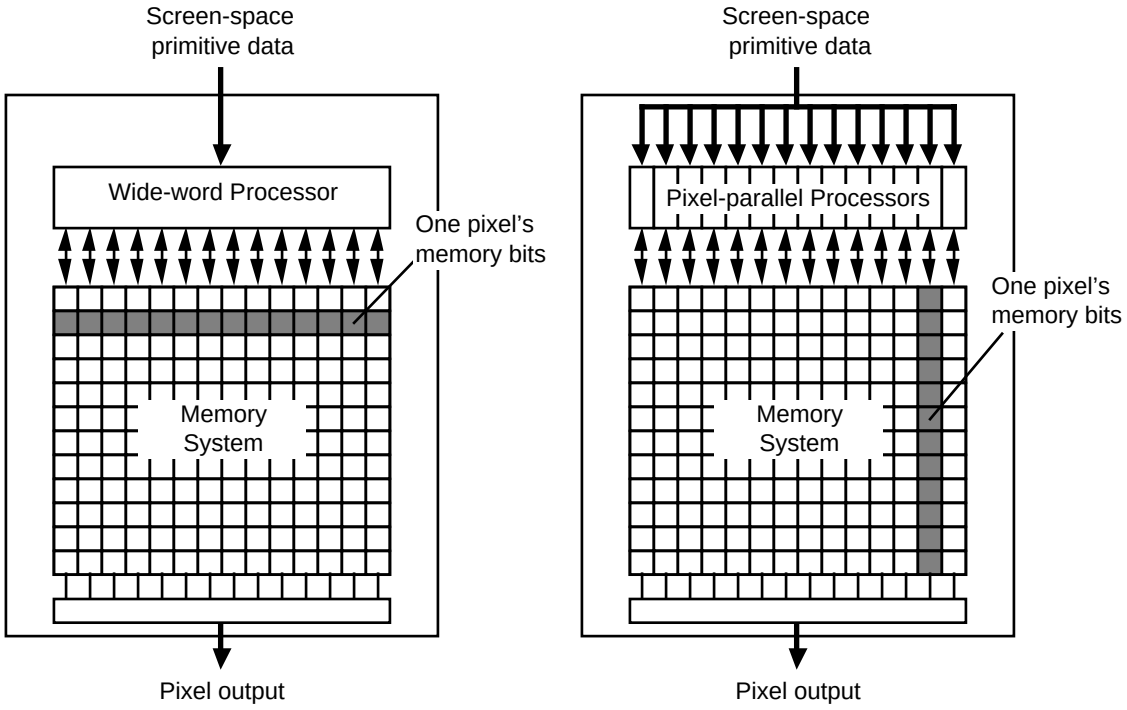
If a screen-subdivision approach is adopted, primitives must be sorted by screen region at the end of the geometry-processing stage. This has the important advantage that the expensive, high-speed pixel storage connected directly to the rasterizer needs to hold only a fraction of the pixels on the screen.

The approach has two disadvantages, however. First, the sorting operation increases latency in the pipeline because geometric transformations (with sorting) and rasterization cannot be overlapped for the primitives in a single frame. This latency may be reduced by performing a quick first pass through the data to perform the screen-region sort. Also, if the scene must be rendered multiple times for supersampling, the increased latency is small compared to the latency of an entire frame.

Second, buffering must be provided for an entire frame's worth of screen-space primitive data, since sorted primitives cannot be used until the sorting is complete. This approximately doubles the storage requirements for the front end. Trading back-end storage for front-end storage is generally favorable, however: sorted primitives can be stored in commodity memories, whereas pixels must be stored (at least temporarily) in expensive high-speed memories, and access rates for primitives are much lower than for pixels.

Processor Organization

The processors on a logic-enhanced memory chip can be implemented in a variety of ways, ranging from a single wide-word processor to numerous single-bit processors. The two extremes of the continuum are shown in Figure 3.



(a) Single, wide-processor organization.

(b) Multiple, narrow processor organization.

Figure 3: Alternate processor organizations for logic-enhanced memories.

In the wide-processor approach, a single processor accesses all the bits of a pixel in parallel. Because the processor handles all of the pixels, its operation must be very efficient. This favors designs with fixed processor functions, carefully tuned for a particular set of rasterization and shading operations. Ideally, the processor can update a pixel in a single cycle. Since there are relatively few processors, each can have its own control store and sequencer, making a MIMD processor organization feasible. Since processors handle pixels sequentially, larger primitives take longer to process than smaller ones. Queueing at the input interface is needed to avoid wasting processor cycles.

In the narrow-processor approach, a processor is provided for each pixel on the chip and all pixels are accessed simultaneously. Because many processors are provided, and each processor updates only a narrow field of pixel memory in a single cycle, the cost of a processor cycle is relatively low. This encourages programmability and flexible allocation of memory bits to function. The narrow-processor approach requires SIMD control, since it is infeasible to provide separate control stores and sequencers for many tiny processors. Since all pixels are processed in parallel, performance is independent of primitive size; it can be guaranteed to match exactly the bandwidth of the primitive input interface.

For scan-conversion calculations, neither approach can take full advantage of the principal leverage for improved speed, the large data bandwidth into on-chip memory. In

the wide-processor approach, the best that can be done is to compute all of the interpolations (*z*, color, surface normals, texture indices) for one pixel in one cycle. If the memory array is wider than the number of bits in one pixel, some of the potential memory bandwidth is lost. It is difficult to build processors wider than 100–200 bits, whereas memory arrays can be as wide as a thousand bits. In the narrow-processor approach, when small primitives are processed on a large array of SIMD processors, most processors correspond to pixels outside the primitive and do no useful work, so only a small fraction of the memory bits available at a given time are actually in use. Neither of these problems is relevant, however, if the rasterizer performance is limited by the bandwidth *into* the chip; that is, if the processors can absorb the full primitive input bandwidth, no further improvements are possible from processor design or memory organization.

For shading calculations, however, the narrow-processor, pixel-parallel approach has a decided advantage. Since shading can be done in parallel for all pixels using deferred shading, a pixel-parallel rasterizer can use virtually 100% of the available compute power and memory bandwidth. A wide-word processor tuned for rasterization would not execute shading algorithms efficiently, since shading calculations do not fit the interpolation/*z*-comparison model of rasterizer calculations.

The tradeoffs between wide-word processors and multiple narrow processors resembles the coarse-grain/fine-grain debates in parallel computing. As in those debates, a compromise between the extremes may prove most effective.

System-Level Issues

So far we have argued that logic-enhanced memories effectively deal with the frame-buffer bandwidth bottleneck. Once this bottleneck is removed, system performance is limited by the single path that feeds screen-space primitives to the rasterizer. This limitation can be circumvented only by applying parallelism *by object*.. This is a large subject in itself[16]. We briefly review here two promising approaches, screen-subdivision and image composition, and show how they affect the design of logic-enhanced memories.

Screen-subdivision. Earlier, we saw that the lack of memory density in ASICs forces primitive sorting by screen region to exploit locality of reference. This sorting has the side benefit of facilitating parallelism by object. If the screen regions for the sort are chosen large enough so that each primitive tends to fall within a single region, multiple streams of primitives can be processed on parallel rasterizers. This idea fundamentally diverges from the single-pipeline model, and enables the design of graphics systems that are parallel by object throughout.

Figure 4 shows a conceptual view of the screen-subdivision approach. Geometry processing (transforming primitives and sorting them by screen region) is performed in parallel. A general-purpose communication network conveys primitives from the

geometry processors to the rasterizers, each of which processes all of the primitives that fall into one or more screen regions for which it is responsible.

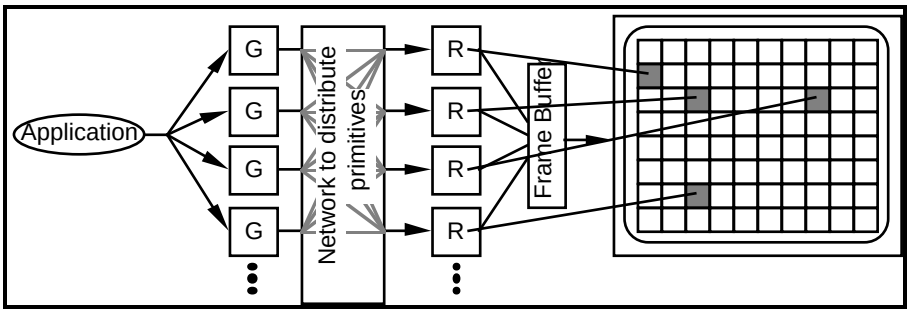


Figure 4: Object-parallelism by screen subdivision. G's are geometry processors and R's are rasterizers.

We explored this approach in Pixel-Planes 5, in which multiple rasterizers, built using logic-enhanced memories, each process pixels of a 128x128-pixel region in parallel. The system's performance is scalable over a wide range by varying the number of rasterizers, and it can render up to 2–3 million Phong-shaded triangles per second. Its logic-enhanced memory chips contain 256 pixel processors, each with 208 bits of memory. An array of 64 of these chips forms a 128x128 pixel processor array with pixel buffer; each rasterizer contains one of these arrays.

This approach has drawbacks, however. First, the network that distributes primitives to the rasterizers carries all of the primitives on every frame. For the machine's performance to scale linearly with hardware cost, the bandwidth of its expensive, fully-connected network must scale linearly with machine size, a difficult proposition. Second, the dynamic assignment of rasterizers to screen regions is difficult to control. Third, static screen subdivision can suffer load imbalances; for example, if all of the primitives fall into one region, all of the speedup from object parallelism is lost.

Image composition. A second way to introduce parallelism by object is image composition. Here, multiple *renderers* each compute a full-screen image of a portion of the primitives. A renderer is essentially a complete graphics pipeline with a geometry processor and a rasterizer. It outputs a stream of pixel data that includes visibility information (*e.g.*, *z* values) as well as colors. A high-speed network composites these images into a final image.

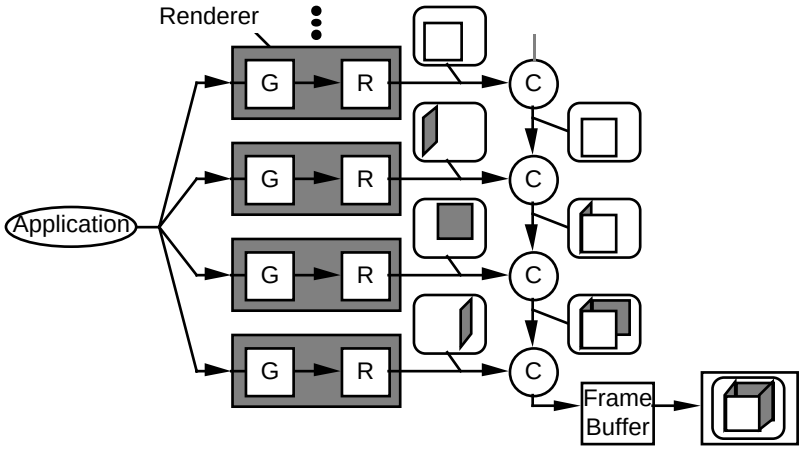


Figure 5: Object parallelism by image composition. G's are geometry processors, R's are rasterizers, and C's are compositor nodes in the image composition network.

Figure 5 is a diagram of the image-composition approach. The approach has two main advantages. First, the image-composition network replaces the primitive distribution network of the screen-subdivision approach; although the image-composition network must be fast, its bandwidth is independent of the number of renderers in the system and so the system scales linearly to arbitrary size, with no obvious limit other than cost. Second, since the renderers are simple graphics pipelines that operate essentially independently, a simple programming model can be employed that hides the parallelism of the system from the user. Its main disadvantage is the cost of the image-composition network, particularly when advanced shading or antialiasing is employed. To use deferred shading, shading must be performed after the subimages are composited, requiring an extra hardware unit or *shader*. Since the intermediate data required for shading is more than simply a color and *z* value per pixel, the image-composition network must have exceedingly high bandwidth.

We have adopted a hybrid of the two approaches for our latest graphics architecture, *PixelFlow*[17], now in design, which will rasterize over 1 million triangles per second per system board and will support high-quality shading, texturing, and antialiasing. PixelFlow contains three types of system boards: renderers, shaders, and frame buffers. Each board includes an array of logic-enhanced memories that contain pixel memory, pixel processors, and special communication circuitry that implements the image-composition network. The final section of this paper describes the PixelFlow Enhanced Memory Chip, as a concrete example of a high-performance logic-enhanced memory design.

PIXELFLOW ENHANCED MEMORY CHIP

The PixelFlow Enhanced Memory Chip (EMC) design exploits advances in CMOS technology and circuit techniques to build compact, high-performance rasterizers that

support advanced shading, texturing and antialiasing. Figure 6 shows a block diagram of the chip. We first describe its memory and processor organization, and then describe enhancements to support real-time image compositing and texture-lookup operations.

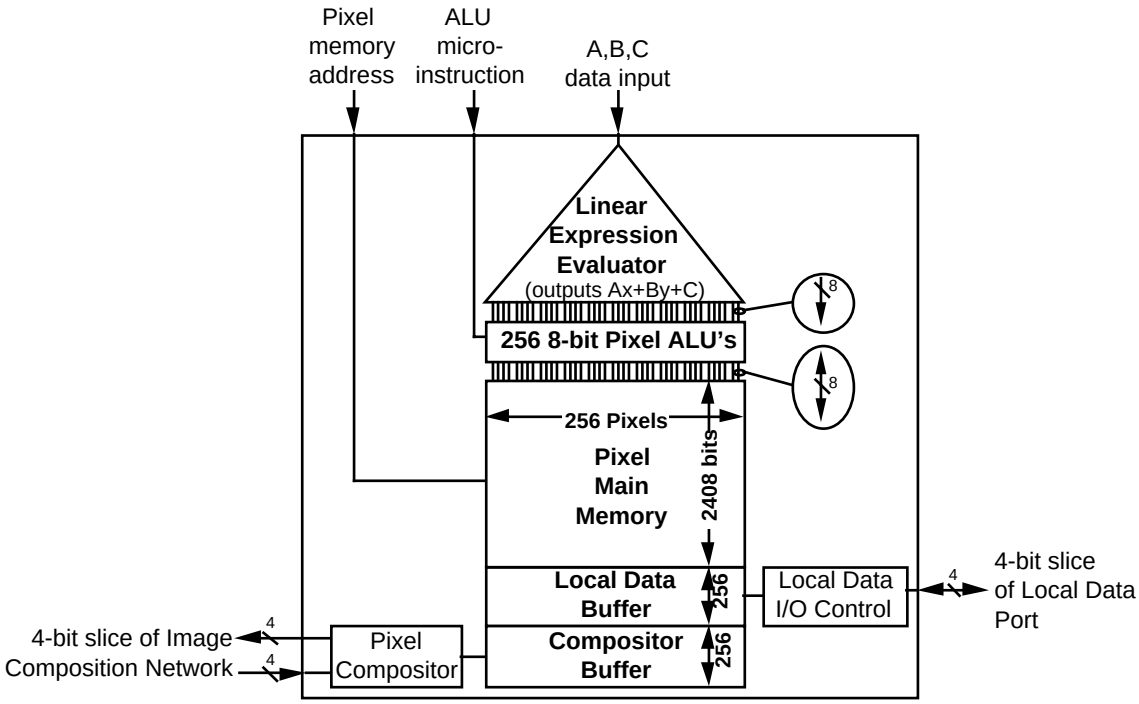


Figure 6: Block diagram of PixelFlow logic-enhanced memory chip.

Memory Organization

Recent work shows that standard logic processes can support dense 1-transistor dynamic memory designs, rather than the 4- or 6-transistor static RAMs normally used in custom ASICs[18]. With a modern, but not aggressive, 0.8-micron CMOS process, 0.5 to 1 Mbit of fast DRAM can be included on a reasonably sized chip. Even such densities do not allow us to provide pixel memory for every pixel on the screen. We therefore use 64 of the EMCs, with 256 pixels per EMC, to build a processor array that can process a 128x128-pixel region in parallel. Each PixelFlow board contains one of these processor arrays.

Since a processor array handles only a 128x128-pixel region of the screen at a time, PixelFlow uses a hybrid approach, with screen subdivision at the board level and image composition between boards, to achieve parallelism by object. On each renderer board primitives are sorted by screen region prior to rasterization, and rendering proceeds by rasterizing one region after another. Once all renderer boards have rasterized a given region, they send intermediate pixel values for that region into the image-composition network. The network delivers composited results to a shader, which operates on regions in pixel-parallel fashion. Shaders also blend multiple samples per pixel for antialiasing.

Once shading is complete, final color values are delivered over the image composition network to a frame buffer.

On renderers, the 2048 bits per pixel of main memory are used to hold pixels for several regions simultaneously. These multiple pixel buffers are used in such a way that rasterization of regions tries to keep ahead of compositing in order to average out load imbalances between regions with few primitives and regions with many. On shaders, the large amount of memory per pixel supports complex shading algorithms that require large amounts of intermediate storage.

Two special 256-bit-per-pixel memory partitions are provided on the EMC, one for each of two I/O ports described below. These partitions can be either joined to main memory and accessed by the pixel processors, or isolated so that I/O operations can be fully overlapped with pixel processing.

Rasterizing Processor Organization

The PixelFlow EMC uses the narrow-processor approach. A simple, but general, 8-bit processor is associated with each pixel's 2560 bits of on-chip memory. The 256 processors on each chip operate in SIMD lockstep: each executes the same instruction on a given cycle, and each accesses pixel data on the same row of memory, specified by a global pixel-memory address. As in most SIMD machines, each processor has an *enable* register that qualifies memory writes. This supports program branching by disabling processors, based on the boolean outcome of a branch test. Though the design is very near the narrow-processor end of the processor-organization continuum, it contains two concessions toward special functions:

- **Linear expression tree.** The EMC contains special hardware for computing interpolated values required for scan conversion. A binary tree of multiply-accumulate nodes computes a linear expression $Ax+By+C$ in parallel for all pixels at screen locations (x, y). A,B,C are equivalent to the forward difference setup parameters broadcast to rasterizing processors in many commercial graphics systems.
- **Hardware support for multiply.** The 8-bit processors contain circuitry to accelerate multiplications, which dominate shading calculations. This small amount of additional circuitry can be viewed as special purpose hardware for shading.

Image Composition Port

Image-composition bandwidth is the product of screen resolution, update rate, and bits per pixel. In PixelFlow, we provide fast hardware for a simple *z*-depth visibility test for image composition, and we perform antialiasing by supersampling. Deferred shading

requires that we composite intermediate pixel values; about 200 bits per pixel are needed to support Phong shading with image-based textures. We have found that reasonable antialiasing can be computed with as few as five samples per pixel. For high-resolution screens at 24 frames per second, the image-composition network must sustain at least 1280 • 1024 • 24 • 200 • 5 = 31 Gbits/second, preferably higher.

This formidable bandwidth is supported in PixelFlow by building the compositor hardware directly onto the EMC. The composition network is bit-sliced across the 64 EMCs, with four single-bit paths on each chip. Connections are point-to-point from chips on one board to chips on a neighboring board. The *z* comparison is performed bit serially, MSB-first, requiring only a small amount of circuitry. The network is clocked at four times the (nominally 66 MHz) system clock rate; these very high speeds are enabled by using low-voltage signaling and on-chip termination[19].

The compositor port operation is flexible so that, in addition to compositing based on *z* values, it can move data from network to EMC memory, from memory to network, and from compositor input port to compositor output port without modification. These additional modes are required to allow pixel data to flow into and out of the shaders, and to bypass busy shaders. They also allow EMC processors to execute general image-composition algorithms (at some sacrifice in speed over the simple *z*-comparison); these algorithms include, for example, A-buffer and multi-pass transparency.

Local Port

Texturing is an example of a general class of algorithm that generates or modifies an image by sampling another image. Texturing requires table lookup operations, which are extremely inefficient on a classic SIMD organization. To support rapid texturing on PixelFlow, the EMC has a local data port that streams data in or out of pixel memory. On shader boards, this port is interfaced to a large texture memory, made from conventional DRAMs, via a separate datapath ASIC that converts nybble-serial data from the port to pixel-wide data and provides buffering. The operation of this port and texture-lookup algorithms will be described in a future publication.

On frame buffer boards, the local port and its associated datapath chips interface to a conventional VRAM frame buffer, which displays the fully composited and shaded images.

Controller

On every cycle, EMCs must be supplied with an instruction for the pixel processors, an address into pixel memory, and a byte of the linear expression coefficients A, B, and C. This is beyond the capability of a general-purpose processor, so a third custom chip, the *Image Generation Controller* (IGC), acts as a controller for the EMC array. The IGC receives a 32-bit word-parallel command stream from the geometry processor, consisting of opcodes and A,B,C coefficient sets. It converts the coefficients into byte-serial, fixed-

point form and sequences EMC operations by broadcasting data, instruction, and address to the EMC array. The IGC also controls operation of the I/O ports.

The IGC contains enhancements that leverage the performance of the front-end processor when antialiasing. Its input interface is designed to be connected to dual ported memory (VRAM) mapped into the geometry processor's address space. The IGC thereby receives its primitive input stream without requiring bus cycles from the geometry processor. A sub-pixel offset register on the IGC allows the multiple samples of the supersampling filter kernel to be computed from a single set of rasterization commands by repeatedly passing these commands from the VRAMs. This greatly improves supersampling performance so that, for example, a single floating-point microprocessor can keep up with the rasterizer when supersampling with 6 or more samples per pixel.

CONCLUSION

We have described how logic-enhanced memories can be used to remove the rasterizer/frame buffer bottleneck that limits the performance of current image-generation architectures. Putting pixel memory on-chip with rasterizing processors provides the 2-3 orders of magnitude improvement in access rates needed to support realistic shading models and antialiasing in interactive systems.

It is neither economical nor necessary to build entire frame buffers with custom memory. Adding a small *pixel buffer* to a rasterizer provides the needed bandwidth improvements. A pixel buffer operates somewhat like a cache for intermediate pixel values, sending only final pixel colors to a conventional frame buffer, built with commodity DRAMs. Because relatively little memory is needed close to the rasterizer, this memory can be added at small incremental cost to the custom silicon universally used for rasterizers in high performance systems.

The design space of logic-enhanced memory rasterizers has several dimensions, including: complexity of processors versus number of processors, amount of memory per processor, and SIMD *vs*. MIMD control. We have argued that the SIMD pixel-parallel approach has many advantages, particularly if deferred shading is employed to support advanced shading and texturing calculations.

Finally, we have shown how the logic-enhanced memory approach can be used to implement object-parallel graphics architectures, based on screen-subdivision and image composition, that provide primitive-per-second performance far above that of current systems as well as perform texturing, advanced shading, and antialiasing in real time.

ACKNOWLEDGEMENTS

This research is supported in part by the Defense Advanced Research Projects Agency, DARPA ISTO Order No. 7510, and the National Science Foundation, Grant No. MIP-9000894.

REFERENCES

[1]. Martin, P., and H. Baeverstad, "TurboVRX: A High-Performance Graphics Workstation Architecture," *Proc. of AUSGRAPH 90*, September 1990, pp. 107-117.

[2]. Silicon Graphics Computer Systems, *Vision Graphics System Architecture*, Mountain View, CA 94039-7311, February 1990.

[3]. Fuchs, H., "Distributing a Visible Surface Algorithm over Multiple Processors," *Proceedings of the ACM Annual Conference*, Oct. 1977, pp. 449–451.

[4]. Clark, J. and M. Hannah, "Distributed Processing in a High-Performance Smart Image Memory," *Lambda (VLSI Design)*, Vol. 1, No. 3, Q4 1980, pp. 40–45.

[5]. Slater, M., "Rambus Unveils Revolutionary Memory Interface," *Microprocessor Report*, Vol. 6, No. 3, March 4, 1992, pp. 15–21.

[6]. Deering, M., S. Winner, B. Schediwy, C. Duffy, and N. Hunt, "The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics," *Computer Graphics* (Proceedings of SIGGRAPH '88), Vol. 22, No. 4, pp. 21–30.

[7]. Ellsworth, D.E. "Parallel Architectures and Algorithms for Real-Time Synthesis of High-Quality Images Using Deferred Shading," Workshop on Algorithms and Parallel VLSI Architectures, Pont-à-Mousson, France, June 12, 1990.

[8]. Pinkham, R., M. Novak, and K. Guttag, "Video RAM Excels at Fast Graphics," *Electronic Design*, Vol. 31, No. 17, 18 Aug. 1983, pp. 161–182.

[9]. Fuchs, H. and J. Poulton, "Pixel-Planes: A VLSI-Oriented Design for a Raster Graphics Engine," *Lambda (VLSI Design)*, Vol. 2, No. 3, Q3 1981, pp. 20–28.

[10]. Eyles, J., J. Austin, H. Fuchs, T. Greer, and J. Poulton, "Pixel-Planes 4: A Summary," *Adv. in Computer Graphics Hardware II* (1987 Eurographics Workshop on Graphics Hardware), Eurographics Seminars, 1988, pp. 183-208.

[11]. Demetrescu, S., "High Speed Image Rasterization Using Scan Line Access Memories," *Proceedings of the 1985 Conference on VLSI*, Rockville MD, Computer Science Press, pp. 221–243.

[12]. Gharachorloo, N., S. Gupta, E. Hokenek, P. Balasubramanian, B. Bogholtz, C. Mathieu, and C. Zoulas, "Subnanosecond Pixel Rendering with Million Transistor Chips," *Computer Graphics* (Proceedings of SIGGRAPH '88), pp. 41–49.

[13]. Fuchs, H., J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel, "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories," *Computer Graphics* (Proceedings of SIGGRAPH '89), Vol. 23, No. 3, pp. 79–88.

[14]. Goris A., B. Fredrickson, and H.L. Baeverstad, Jr., "A Configurable Pixel Cache for Fast Image Generation," *CG&A*, Vol. 7, No. 3, March 1987, pp. 24–32.

[15]. Apgar, B., B. Bersack, and A. Mammen, "A Display System for the Stellar Graphics Supercomputer Model GS1000," *Computer Graphics* (Proceedings of SIGGRAPH '88), Vol. 22, No. 4, pp. 255–262.

[16]. Foley, J.D., A. van Dam, S.K. Feiner, and J.F. Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley, Reading, MA, 1990 (especially Chapter 18, "Advanced Raster Graphics Architecture").

[17]. Molnar, S.E., J.G. Eyles, and J.W. Poulton, " PixelFlow: High-Speed Rendering Using Image Composition," to appear in *Computer Graphics* (Proceedings of SIGGRAPH '92), Vol. 26, No. 3.

[18]. Speck, D., "The Mosaic Fast 512K Scalable CMOS DRAM," *Proceedings of the 1991 University of California at Santa Cruz Conference on Advanced Research in VLSI*, 1991, pp. 229–244.

[19]. Knight, T. and A. Krimm, "A Self-Terminating Low-Voltage Swing CMOS Output Driver," *IEEE Journal of Solid-State Circuits*, Vol. 23, No. 2, April 1988, pp. 457–464.

John Poulton is a research associate professor of computer science at the University of North Carolina at Chapel Hill. He received a BS in physics from Virginia Tech in 1967, MS in physics from SUNY-Stony Brook in 1969, and PhD in physics from UNC-Chapel Hill in 1980. Current research interests include high-performance computer graphics systems and VLSI system design and prototyping. He is co-PI (with H. Fuchs) of the Pixel-Planes research project.

John Eyles is a research assistant professor of computer science at the University of North Carolina at Chapel Hill. He received a BS in mathematics from Davidson College in 1974, MS in biomedical engineering and mathematics from UNC-Chapel Hill in 1980, and PhD in biomedical engineering and mathematics from UNC-Chapel Hill in 1982. Current research interests include enabling technologies for virtual environments, especially high-performance graphics architectures and real-time head tracking, and rapid prototyping of VLSI-based systems.

Steven Molnar is a research assistant professor of computer science at the University of North Carolina at Chapel Hill. He received a BS in electrical engineering from Caltech in 1986 and an MS and PhD in computer science from UNC-Chapel Hill in 1988 and 1991. Current research interests include architectures and algorithms for real-time, realistic image-generation and VLSI-based system design.

Henry Fuchs is Federico Gil professor of computer science and adjunct professor of radiation oncology at the University of North Carolina at Chapel Hill. He received a BA in information and computer science from the University of California at Santa Cruz in 1970 and a PhD in computer science from the University of Utah in 1975. He serves on advisory committees, including that of NSF's Division of Microelectronic Information Processing Systems and ShoGraphics' Technical Advisory Board. Current research interests include high-performance graphics hardware, 3D medical imaging, and head-mounted display and virtual environments.

Authors can be reached at the Department of Computer Science, Sitterson Hall, University of North Carolina, Chapel Hill, NC 27599-3175, USA.