To appear in "Proceedings of the NATO Advanced Study Institute on Microarchitecture of VLSI Computers", Sogesta-Urbino, Italy, July 9-20, 1984

VLSI Architectures for Computer Graphics[1]

Gregory D. Abram
Henry Fuchs

Department of Computer Science
University of North Carolina at Chapel Hill, USA

Abstract

Both academic researchers and commercial concerns are increasingly interested in applying VLSI technologies to graphics systems:

- For researchers, graphics systems offer an attractive model for study of computer architectures in VLSI: these systems have a small well-defined set of operations and simple data and control structures, making these systems ripe for applying parallelism and modularization techniques; many of these systems, especially the interactive high-resolution color ones, have severe computation demands that are unfulfilled by solutions embodied in current systems.

- For commercial concerns, there is a rapidly increasing market for interactive graphics systems as personal workstations in which graphics displays replace text-only terminals.

In this paper, we cover: a) the conceptual organization of a "generic" graphics system and its realization in several state-of-the-art commercial products; b) the architecture of several recent VLSI chips and systems and their likely effect on the organization of future graphics systems; c) the architecture of several VLSI-based systems that are currently subjects of research. The design strategies used in these systems -- the structure of parallelism, intertwining of data and computation, the tradeoff between custom and off-the-shelf parts -- may provide insights into other applications as well.

1. INTRODUCTION

The design of graphics systems has been a challenging topic of study for several decades; the demands for ever-increasing performance have always pushed the available technology to its limits. The availability of off-the-shelf TTL circuitry in the early 1970's allowed custom designs of minicomputer-level complexity. The advent of large-scale RAMs allowed systems to store complete images and quickly and randomly address any

[1] This research was supported in part by the (USA) National Science Foundation under grant ECS-8300970 and by the Defense Advanced Research Projects Agency contract DAAG 29-83-K-0148.

pixel in them; this capability gave rise to the current boom in color raster systems. Inexpensive microprocessors allowed these frame buffer systems to perform many functions independently of the host computer.

The possibility of custom VLSI promises another level of power in affordable graphics systems. The increased plasticity of custom VLSI allows systems designed using this medium to take on radically different structures than seen heretofore. This paper explores some of these possible structures -- a few just recently announced, most yet to come.

We concentrate in this paper on interactive color raster systems aimed at laboratory or office use, mostly for 3D applications; a few related systems that focus on 2D applications are included [Gupta, Sproull, et. al., 1981]. Of course, most of the systems cited can be used for a wide variety of applications, not restricted to 3D. We have intentionally left out systems aimed at the expensive ($1M) flight simulator market, largely due to lack of available information in the public domain [Schachter, 1981], although a number of the systems covered in this paper may be used for flight simulator applications.

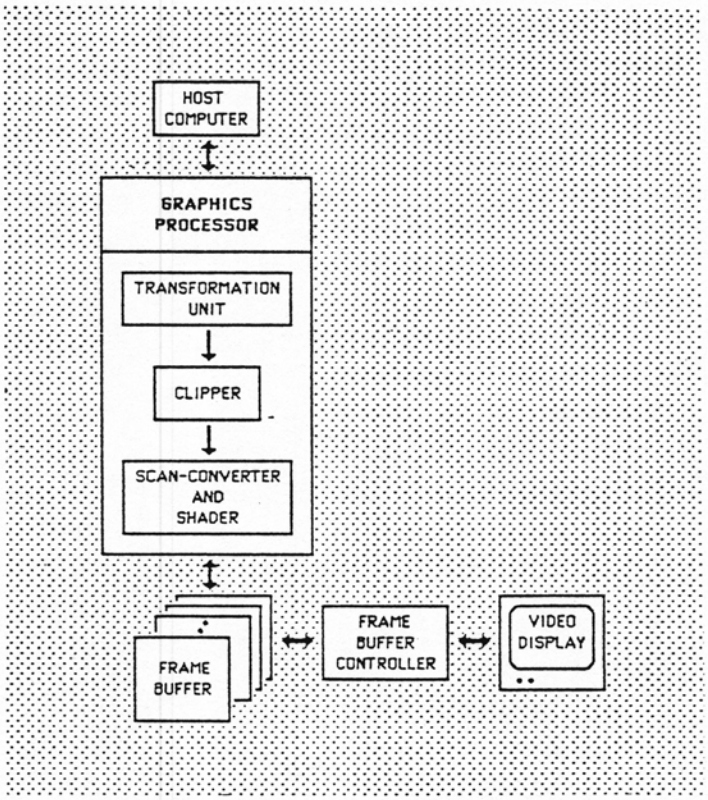

Figure 1: A Typical 3D Raster System Organization

The overall organization of many raster graphic display systems is quite similar (Fig. 1). The central feature is the frame buffer memory in which is stored the image currently being displayed -- and perhaps one or more additional images. To relieve the host computer from low-level tasks, one or more processors are attached to most frame buffers. The nature and organization of these processors is one of the major focuses of

this paper. Its major tasks for 3D image generation are illustrated in Figure 2. An alternate organization is used for many general purpose workstations, in which the image usually shows one or more pages of mostly textual information (rather than an interactive 3D image). The major tasks involve generation and movement of 2D image data; the typical hardware organization for such systems is shown in Figure 3.

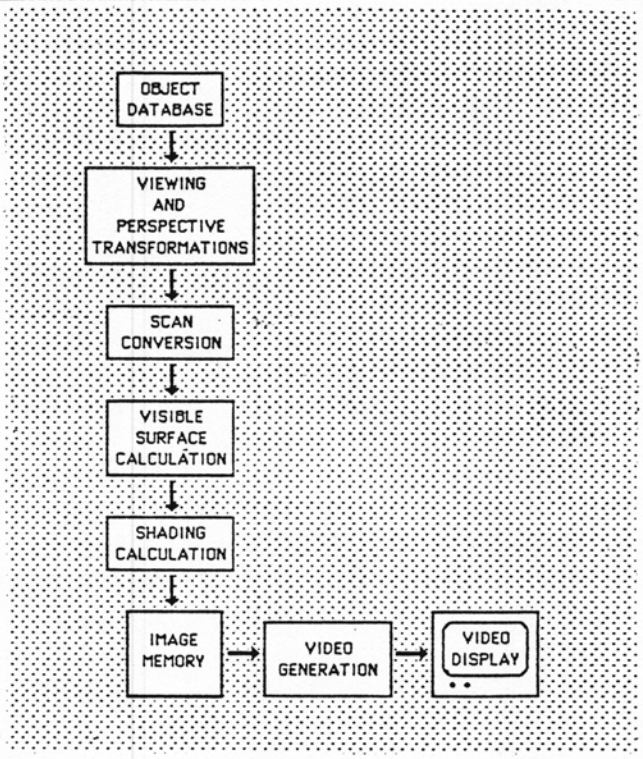

Figure 2: Functional Steps for 3D Image Synthesis

In studying many graphic system designs currently being developed, several distinct strategies become evident.

- Implementing innermost loops in hardware

One obviously reasonable strategy to consider is to transfer an often-executed inner loop from software to hardware [Atwood, 1984; T. Ikedo, 1984]. The details of these designs are discussed in section 2 . As some have noted, however, this strategy may not always succeed [Pike, 1984]. The new hardware inner loop solver may add so much more overhead as to swamp any gains it produces in solving the inner loop faster.

- Integrating a boardful of functions onto a single chip

Some systems have succeeded by restructuring an extant solution into VLSI components and thereby reducing a module that formerly needed one or more boards of parts to a few custom chips [Clark, 1982]. As will be seen below, however, it is not

always obvious how to restructure the board-level function in such a way as to enable a VLSI-based solution.

- Alternative architectures

The restructuring for a VLSI-based solution can extend beyond the board to system level; with custom VLSI, it is appealing to attempt a radical restructuring of the problem in hopes of achieving a solution that's much more attractive in this new medium [Fuchs and Poulton, 1981; Fuchs, Poulton, et. al., 1982; Kedem and Ellis, 1984].

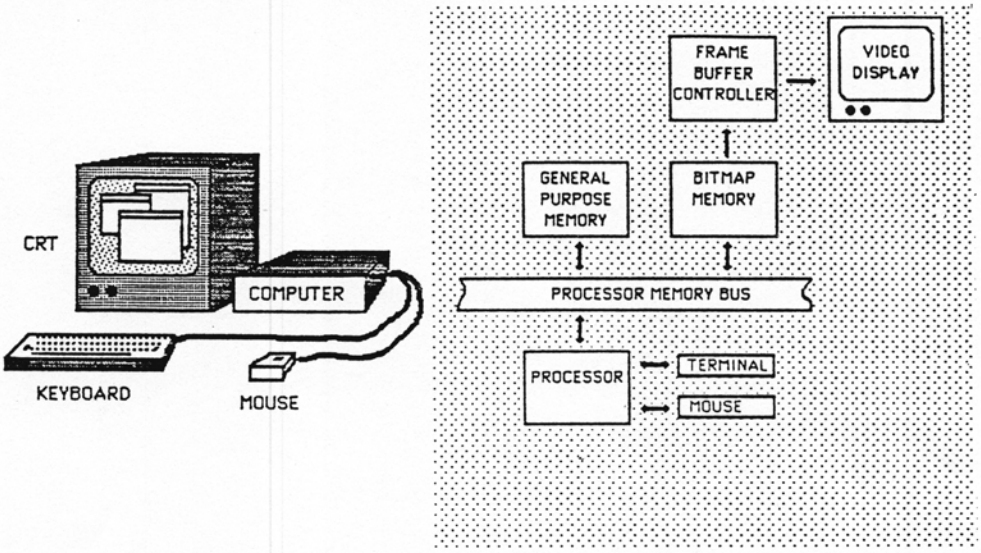

Figure 3: A Bitmap Display and Typical Hardware Organization

In this paper, we review several current and several proposed systems which take advantage of VLSI technology. Associated with each will be a figure noting the structure of such a system. It should be kept in mind that these architectural layouts are conceptual models only, reflecting our own understandings, and may bear only superficial resemblance to actual implementations.

2. HARDWARE FOR CRITICAL LOW-LEVEL FUNCTIONS

One characteristic of computer graphics systems is that a few low-level functions are used extremely frequently and, therefore, account for large portions of the total work done. Much work has been done to speed up the software algorithms used to perform these functions. Major advancements may be achieved by supporting these functions in hardware. Two such functions are the line drawing algorithm, which determines which pixels best approximate a line, and the "raster op", a complex function which allows logical functions between arbitrary rectangular regions of a bitmap display [Bechtolsheim and Baskett, 1980; Thacker and McCreight, 1979].

2.1 VLSI Support for Line Drawing

Although major strides have been made in the design of raster graphics systems, random-scan (also called vector or calligraphic) systems remain the technology of choice for line-drawing applications. This is for two reasons: image generation time and image quality. Raster systems must compute the set of pixels which best approximate lines and set them accordingly; random-scan systems use analog circuitry to drag the electron beam across the CRT screen from endpoint to endpoint. Since raster systems have only a relatively coarse grid of addressable pixels, images show distracting staircasing effects along edges (unless costly anti-aliasing algorithms are used); in contrast, lines on random-scan systems are smooth.

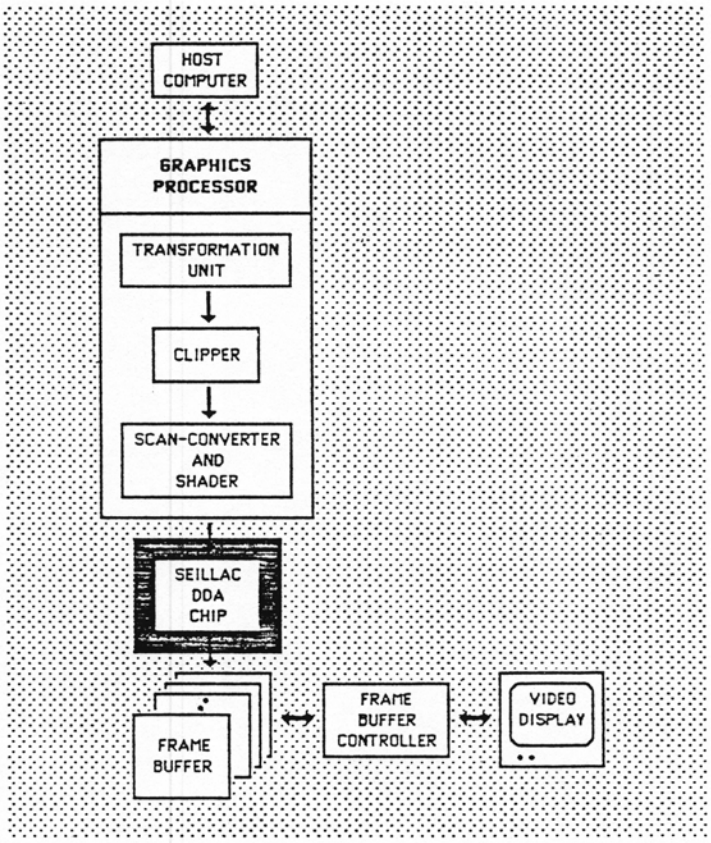

Figure 4: Organization of System Using SEILLAC DDA Chip

The SEILLAC-7, a new graphics system built by the Seillac Co., Ltd., utilizes a custom ECL DDA chip to achieve extremely high line drawing rates (Fig. 4). It is claimed to be about five times faster than previous raster systems that lack such special-purpose hardware (though other systems claim similar speeds, such as the Ramtek 2020) [Ikedo, 1984]. This chip, which achieves a speed of about 40 nanoseconds per pixel in the line, includes a function to modulate the pixel intensity to alleviate the staircasing effects. In doing so, the images generated are claimed to approach stroke-drawn systems both in image quality and in vector drawing speeds.

2.2 Bitmap Manipulations

A recent development in professional workstations has replaced the standard ASCII terminal with a high resolution black-and-white frame buffer system (a "bitmap" display). This approach, pioneered in the Xerox Alto system in the early 1970's [Thacker, McCreight, et. al., 1971], offers many advantages over standard ASCII terminals; for example, high quality graphics and arbitrary fonts can be used for document preparation.

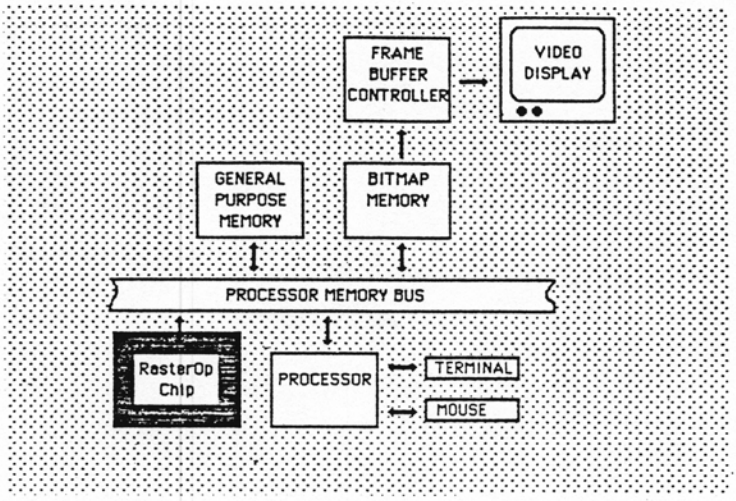

Figure 5: Bitmap System Organization Using RasterOp Chip

Bitmap displays require that the graphics system perform operations (Raster Ops) on bitmap memories efficiently (Fig. 5). Scrolling a bitmap window requires that the window be copied up one line; this must be done very quickly if the display is to be useful. Raster Ops typically allow logical operations on source and destination bitmaps; the copying process uses the function:

$$f(\text{source, destination}) = \text{source}$$

whereas the function:

$$f(\text{font, destination}) = \sim \text{font}$$

may be used to write reverse video characters.

The implementation of the RasterOp function, however, is tricky. First, source and destination areas may overlap; the algorithm must to careful to operate in an order which ensures that data will not be overwritten before it is used. For example, if the destination is to the *left* of the source, the operation must proceed from left to right across the source, whereas if the destination is to the *right* of the source, the pass must be in the opposite direction.

The problem is further complicated by the organization of the bitmap memory. Bitmap displays are often organized with 16 or 32 horizontally adjacent pixels in a single word. Since regions do not necessarily fall on word boundaries, corresponding pixels in

source and destination words may fall at different bit positions within the words. In order to operate on the several pixels within each memory word in parallel, *two* source words must be available to be to aligned with the data within the destination data word. The logical operation is then applied to the aligned words, and the result written to the destination location. This must be repeated for each word which contains a destination pixel.

Silicon Compilers, Inc., in conjunction with Sun Microsystems, Inc., have implemented a chip to support the RasterOp function [Iannamico and Atwood, 1984]. This chip utilizes a two word FIFO to manage the source data words; the adjacent words are fed to a barrel shifter to align them with the destination data. Alternately, a pattern register is available for repetitive source data (for example, if a background pattern is to be written). The data words are fed to a simple ALU to compute the logical operation; a function decoder allows the host to choose among 8 possible functions of pattern, source, and destination pixel values. Finally, mask registers can be used to protect bits of the destination data words which lie outside the destination area.

A substantial amount of work remains for an external controller to do (either the host CPU or an external finite state machine). Unlike the Seillac DDA chip outlined above, the looping here must be handled by the controller. The chip has no memory-addressing capability; it must rely on the controller to spoon-feed it input data and to return the results to the destination memory locations. The chip does, however, provide functions which may be costly for conventional microprocessors, including in particular the arbitrary 32-bit shift necessary for data alignment.

3. INTEGRATED SYSTEM COMPONENTS

Whereas greater performance can be had by supporting critical functions in hardware, both performance and cost can be addressed by integrating large parts of the conventional graphics system. Some functions typically built out of large numbers of chips can, in fact, be implemented directly on a single (or a very few) VLSI ICs. This substantially decreases the chip count and can greatly improve the performance.

3.1 TI 4161 Memory Chip

Two related problems plague the frame buffer memory designers: 1) contention between image generation and scan-out for memory access, and 2) the high part count (and associated cost) of satisfactory designs (Fig. 6). For a 1024x1024x1 system refreshed at 60 Hz., a pixel (ie. one bit of the frame buffer memory) must be available for display every 16 nanoseconds (or less). This rate can be achieved by interleaving the pixel memory among several memory chips, which are read in parallel into a high speed shift register which then shifts pixels out at the desired rate. Assuming that the frame buffer is built using 64Kx1 RAM chips, 16 chips are required for a 1024x1024 bitplane, and scanout requires a memory cycle every 16x16 = 256 nanoseconds, leaving little for image generation (unless very high speed--and therefore expensive--memories are used). Using

Possible Memory Organizations		
memory chip size	desired access rate	data path width
256Kx1	64nsec.	4
64Kx1	256nsec.	16
16Kx4	1024nsec.	64

16Kx4 RAMs, we use the same number of chips and get 64 pixels in parallel, and we need a memory cycle every 1024 nanoseconds. Unfortunately, we now need four times as many memory chips and the data path is four times as wide. (Also, with these 4-bit wide chips, modifying a single bit is often awkward, necessitating a read-modify-write operation.)

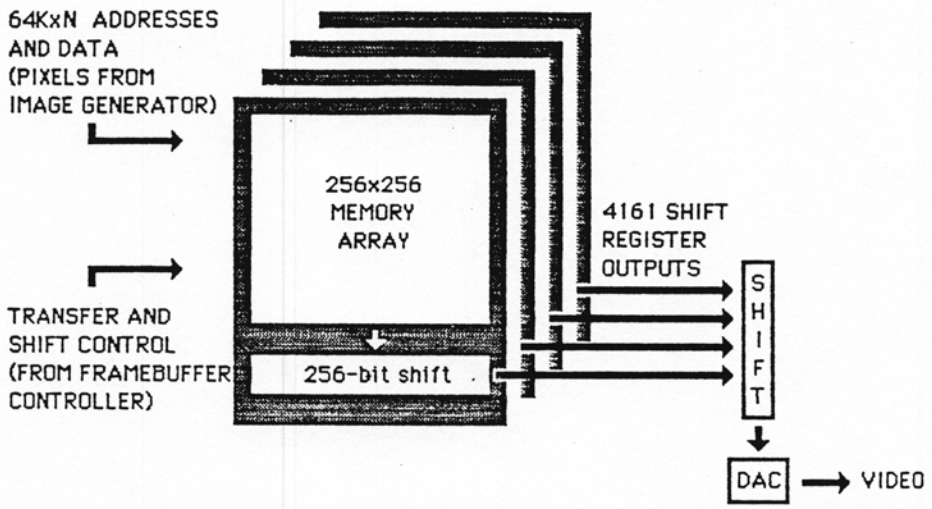

Figure 6: A Possible Frame Buffer Design Using TI 4161 RAMs

In other words, achieving the necessary data rates requires using small-capacity RAMs; whereas achieving low parts counts requires using large-capacity RAMs.

Texas Instruments has recently brought to market a special dynamic memory chip to help solve this problem [Pinkham, Novak, et. al., 1983]. Much like conventional 64K RAMs, the TI 4161 memory is organized as 256 rows of 256 columns. The difference is that a 256-bit 40 nanosecond shift register is included. A command causes an *entire row* to be transferred to the shift register; the chip then acts as two completely independent chips: the 256-bit shift register and a normal 64Kx1 DRAM. In this manner, the chip allows a low system parts count while not tying up memory for scan-out (one memory cycle accesses 256 pixels).

3.2 The Geometry Engine

Three-dimensional image generation requires that each coordinate in the scene be transformed from a object space coordinate system to an image space system and then clipped to the visible region. Because these functions are time-consuming (including multiplies and divides) and lie in the critical data path, this was one of the first image generation components to be implemented directly in hardware.

James Clark, of Stanford University and Silicon Graphics, Inc., has implemented a chip (the Geometry Engine) which, when organized in a twelve-stage pipeline, performs three-dimensional viewing transformations, a perspective transformation and clipping (Fig. 7) [Clark, 1982]. It achieves a rate of 65,000 coordinate points per second and (unlike most earlier systems) operates on floating point numbers. Because a relatively small number of identical ICs are used to implement this pipeline, the cost is low, The

cost will be lower still when multiple copies of the present IC will fit onto a single die.

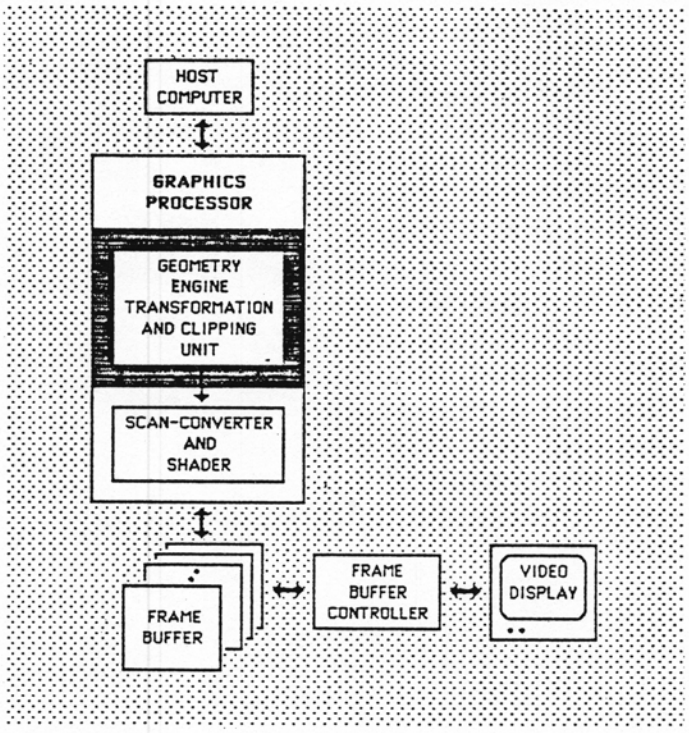

Figure 7: System with Geometry Engine for Transformation and Clipping

3.3 Graphics Display Controllers

Possibly the first specialized integrated circuits for video generation were single-chip video sync generators. Even with such chips, graphics display controllers typically require large amounts of logic, and hence are quite expensive to build out of off-the-shelf MSI and SSI components. Lately, however, two VLSI graphics display controllers have entered the market, each designed for a specific corner of the graphics market.

The NEC 7220 (second sourced as the Intel 82720) is designed to handle high-resolution (1024x1024) color raster graphics systems (such as the Vectrix VX384, a 670x480 system with 9 bits per pixel). The 7220 sits between the host processor (often an Intel micro) and the video memory (Fig. 8). Its video generation circuitry provides a great deal of flexibility, including provisions for zooming, panning, and windowing the image, plus the ability to use a light-pen input device. It also supports image generation by on-board line, arc, area fill and other graphic primitive display functions. Using the 7220, a complete high quality graphics system can be added to a microprocessor system at little more than the cost of the memories and the controller itself.

In contrast, the TI TMS9118 family of graphics display controllers are aimed at the low-cost world of video games, requiring only three chips to add graphics capability onto a standard microprocessor (Fig. 9) [Williamson and Rickert, 1983]. Although they allow only low resolution 256x192 images, they contain support for several specialized functions,

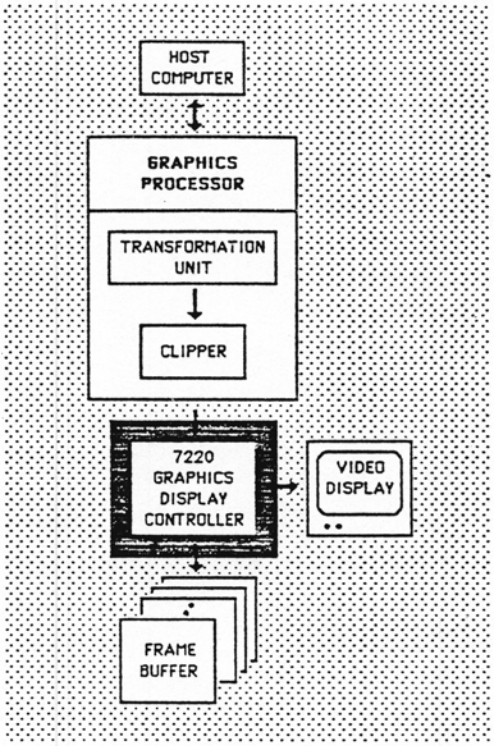

Figure 8: Graphics System Using 7220/82750 Graphics Display Controller

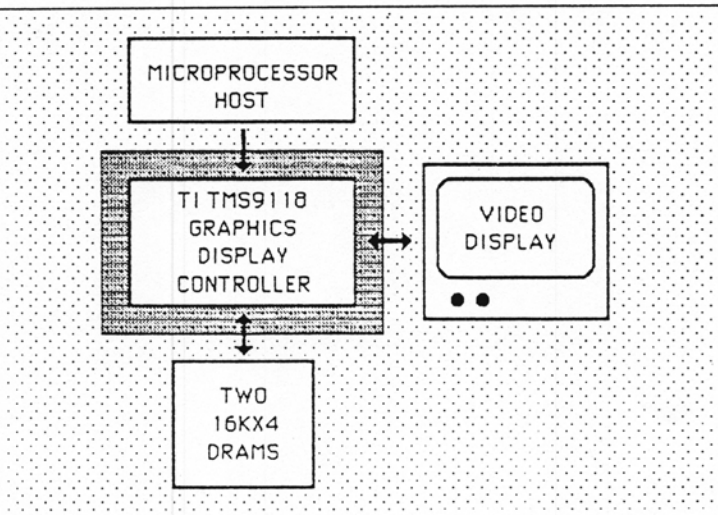

Figure 9: Low-cost System Using TI TMS9118 Graphics Display Controller

including 32 so-called "sprites". A sprite is a small object defined by a rectangular grid of pixels whose position on the screen can be set by simply storing its location in a register, rather than actually copying its pixels from place to place in a (full-image) frame buffer. By using sprites for moving objects in the display, even extremely low-cost devices can support certain classes of very high quality interaction.

4. ALTERNATIVE ARCHITECTURES FOR VLSI

Several current research projects are investigating ways to restructure the traditional graphics architecture to take better advantage of VLSI technology; in particular, the capability of applying potentially many specialized processors to the problems of image generation. These alternative architectures divide into two classes: those that divide the problem in image space and those that divide the problem in object space. Image space strategies divide the *image plane* into independent subsets and associate a separate processor to each. Object space strategies instead divide the *object database* and assign a processor to each.

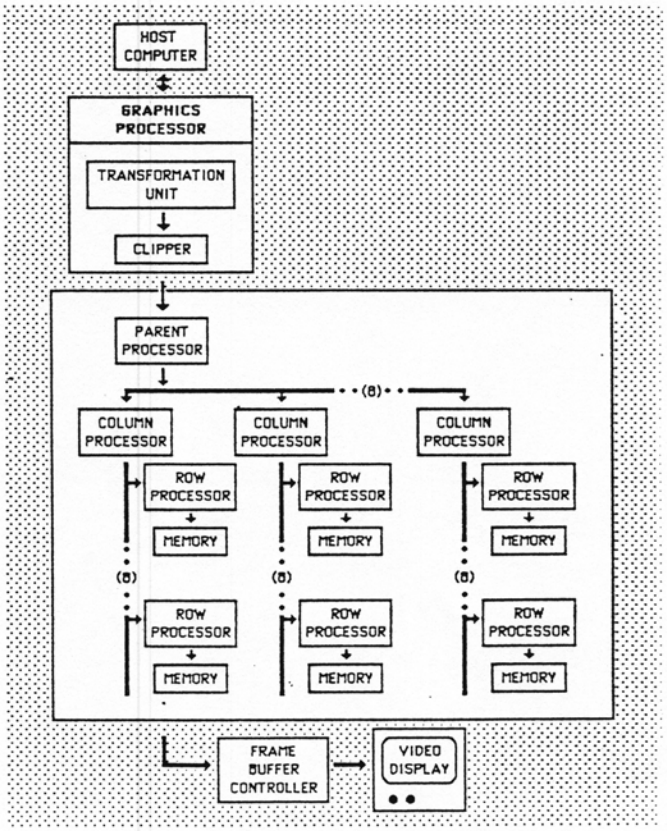

Figure 10: Clark and Hannah 8x8 Display Architecture

4.1 Image Space Strategies

A hard constraint on the performance of raster systems is the bandwidth into the frame buffer memories. However, we can increase the bandwidth to the memories by splitting the image memory into separate components (on the image generation side; the memory still should look contiguous to the scan-out hardware). By associating pixel generating power with each component, these separate components can be accessed in parallel, effectively multiplying the bandwidth to the memories by the number of separate components. In this section, we look at two such strategies. (For speeding up the restricted case of images composed solely of axis-oriented, filled rectangles, see [Whelan, 1982].)

4.1.1 Clark and Hannah James Clark and Marc Hannah have proposed an image space segmentation approach which splits the image screen into segments the size of the RAM chips used (Fig. 10) [Clark and Hannah, 1980]. This organization is similar to an earlier system in [Fuchs and Johnson, 1979]. Both these systems distribute the image buffer in an interlaced fashion in X and Y among many small memories, each controlled by a small processor. For example, a 1024x1024 bit plane can be built out of 64 16K RAMs. These RAMs are interlaced so that for any 8x8 area of the bit plane, one bit comes from each of the these systems RAM chips. Thus, each memory contains every eighth pixel in every eighth row.

Clark and Hannah's system contains an intermediate layer of "column" processors between the main "parent" processor and the memory controlling "row" processors. Each of these processors then does a share of the image generation computation. To generate a line, the parent processor first determines the starting column, slope, line width and ending column of the line, and transfers this information to the column processors. The column processors then determine the part of the line intersecting the associated column of the image memory and transfers this to the row processors. Finally, the row processors actually write pixels into the image memory.

4.1.2 Pixel-Planes We, together with colleagues A. Paeth and A. Bell at Xerox Palo Alto Research Center, have been working on an image-generating system, "Pixel-planes" that performs low level pixel operations within "smart" custom memory chips that make up the frame buffer (Fig. 11) [Fuchs and Poulton, 1981; Fuchs, Poulton, et. al., 1982]. The memory chips autonomously perform, 1) scan conversion (calculating the pixels that fall within a line-segment, convex polygon, or circle), 2) visibility calculations based on the depth ("Z") buffer algorithm, and 3) pixel painting (either "flat" or a limited Gouraud smooth shading).

Efficient implementation is possible because each of the above operations can be performed by variations of the same calculation at every pixel, $F(x,y)=Ax+By+C$ where x,y is the address of the pixel. This function can be efficiently realized on silicon by a complete binary tree with a pixel at each terminal node and a one-bit adder paired with a one-bit delay at each non-terminal node. This circuitry and the other needed processing circuitry (a one-bit ALU at each pixel) is sufficiently compact so that the area of the chips consist of half standard memory cells and half the processing circuitry described above.

Since both shading and depth can be formulated in similar equations, Pixel-Planes based systems can perform Gouraud-like smooth shading and Z-buffer visible surface computations. Two working prototypes have been built at UNC; the latest prototype's chips each contain 2K bits of memory distributed among 64 pixel processors, each with 32 bits of memory. Based on conservative speed estimates, (10 MHz clock), the system is expected to process 25,000 to 30,000 arbitrarily-sized polygons per second.

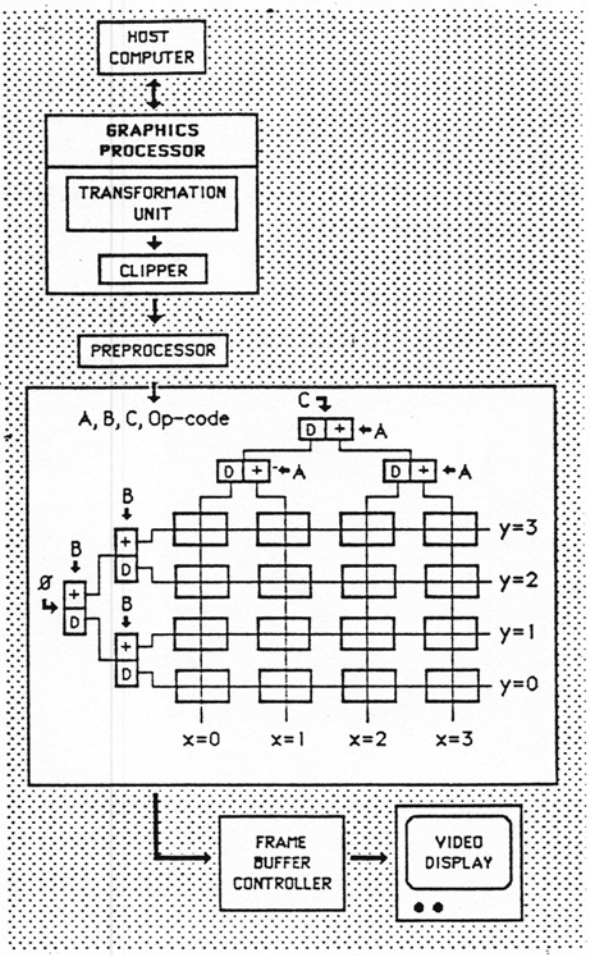

Figure 11: A Pixel-Planes System (4x4 resolution)

4.2 Object Space Subdivision Approaches

An alternative opportunity for parallel processing in image generation is to subdivide the *input data*, assigning separate hardware to each subdivision. Some of the earliest real-time flight simulation systems used this approach; unfortunately, at the time hardware had to be built out of a large number of simple parts, and was therefore extremely expensive and limited in scope [Schumacker, Brand, et. al., 1969]. Using VLSI technology, however, small, specialized processors can be built to perform the necessary operations. Several new designs have been proposed along these lines.

4.2.1 Gershon Kedem's CSG Machine Gershon Kedem has proposed an architecture for the display of objects defined using Constructive Solid Geometry (CSG) (Fig. 12) [Kedem and Ellis, 1984]. CSG is a strategy for computer-aided design in which designs consist of several primitive shapes (spheres, cones, prisms etc.) which are combined using regularized set operations (UNION, INTERSECTION, ADDITION and SUBTRACTION). CSG structures are very naturally represented as binary trees in which leaf nodes

correspond to primitives and internal nodes correspond to the operation which combines the objects described in the two subtrees.

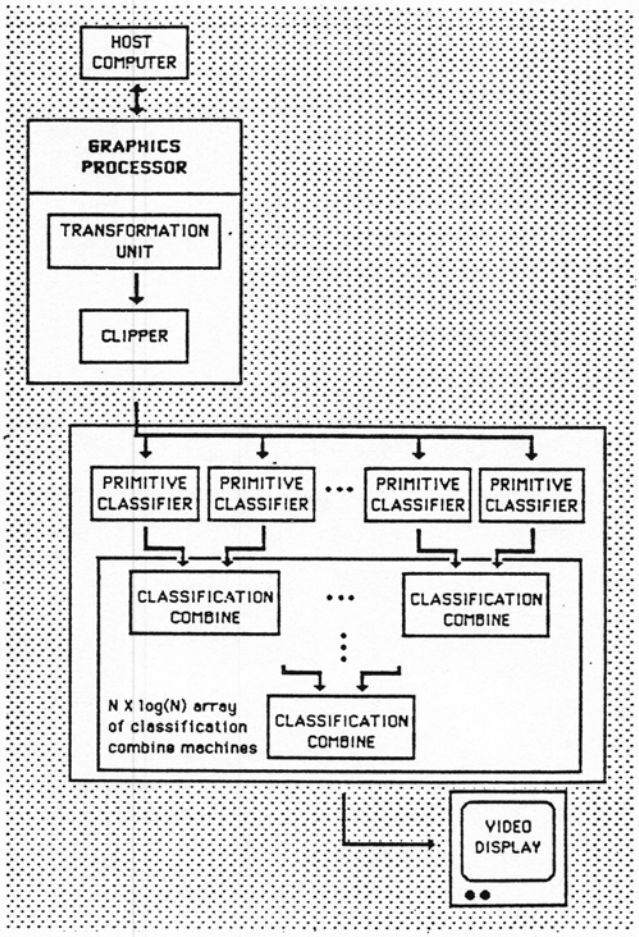

Figure 12: Kedem's CSG Machine Architecture

Kedem's approach instantiates the CSG tree directly in hardware. A reconfigurable tree structure is built which consists of two types of nodes: Primitive Classifiers (PCs), for leaf nodes, and Combine Classifiers (CCs), for internal nodes. To compute a pixel value, the PCs compute (in parallel) the intersections of the ray rooted at the eye point and passing through the pixel center with their associated primitive objects. These intersections (actually line segments of the ray) filter up the tree. Each CC takes the line segments of its left and right subchild and applies its operator on them and passes the result up the tree. The final result, produced at the root of the tree, is then used to compute a pixel shade.

4.2.2 Cohen and Demetrescu Cohen and Demetrescu, in [Cohen and Demetrescu, 1980] have proposed a system that assigns a processor to each potentially visible polygon in the image space (i.e., already transformed world model polygon) (Fig. 13). These, processors are connected as a pipeline and are operated in synchrony with the video genera-

tion. For each pixel on the screen, a token is passed through the pipeline of polygon processors. This token carries the shade and depth of the closest point found for this pixel. This depth is the distance from the viewing position of the closest polygon encountered at this pixel; thus the shade is the best guess so far of the color seen at this pixel. Each processor in turn tests whether the pixel lies inside its polygon. If the point lies inside, the processor compares this depth with its polygon's depth at this point. If the polygon's depth is closer, its depth and color replace the token's data. For real-time image generation, tokens pass in raster-scan order and travel at video rates; that is, each processor must make each decision in one pixel time.

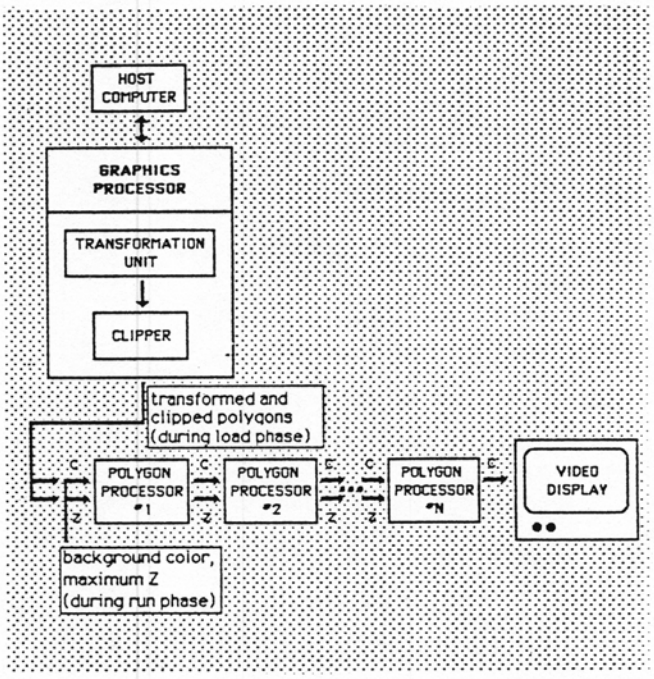

Figure 13: Cohen and Demetrescu's Pipelined Architecture

An elegant feature of this approach is that the pixels stream out of the end of the pipeline in raster-scan order and each value represents the color of the nearest polygon at that pixel; thus the data can be routed directly onto a video display screen.

Weinberg, in [Weinberg, 1981], proposes an elaboration on this design which addresses the problem of anti-aliasing by passing multiple depth-sorted tokens for each pixel along with subpixel masks. Each processor then determines the portion of the pixel covered by its polygon, and compares with the token's mask when the correct position in the depth order is found. If preceding polygons do not completely obscure it, it is added to the token chain. Subsequent tokens are then examined to see if the new polygon completely obscures them, deleting those that are. A filter section at the output uses this data to determine an output shade for each pixel.

This general approach features great modularity; it consists of identical processors hung together in a simple pipeline. It is easily expandable by simply adding more proces-

sor chips. The design costs are held down by the fact that only a single IC needs to be developed; manufacturing costs are held down by the simple structure. The only difficulties may be 1) implementing enough processors so that there is one for each and every polygon in the most complex scene in the intended application, and 2) making each processor sufficiently fast to complete all its calculations for a pixel in one pixel time.

5. SUMMARY AND CONCLUSIONS

As the reader is likely to gather from the above list of designs, we are currently witnessing a blossoming of creative designs for harnessing a new medium to solve an old problem. The good news in all this is that with all this attention, there is likely to be substantial progress; indeed, virtually with each passing month a new system with increased performance is introduced –usually found to contain some custom integrated circuitry. In the next few years, many of the designs described above will, no doubt, be developed, refined, and tested. The effective ones will be adopted, the others will be improved or abandoned. Further in the future, we may see designs that integrate even more of the display system functions – perhaps including the display itself within the processing and image memory. We can then look forward to carrying around a display the size of a book, whose surface is a high-resolution display with built-in high-speed image generating capabilities – thus approaching the predictions of visionaries' "dynabooks" [Kay, 1977] and eye-glass mounted "ultimate displays" [Sutherland, 1965].

References

Atwood, J. "Raster-Op Chip Overview", Silicon Compilers, Inc., Los Gatos, California (1984).

Bechtolsheim, A. and F. Baskett, "High-Performance Raster Graphics for Microcomputer Systems", *Computer Graphics (SIGGRAPH '80 Proceedings)*, Vol. 14, No. 3, (July 1980) 43-47.

Clark, J. and M. Hannah, "Distributed Processing in a High-Performance Smart Image Memory", *VLSI Design*, Vol. I, No. 3, 4th. Quarter, (1980).

Clark, J. "The Geometry Engine: A VLSI Geometry System for Graphics", *Computer Graphics (SIGGRAPH '82 Proceedings)*, Vol. 16, No. 3, (July 1982) 127-133.

Cohen, D. and S. Demetrescu, Presentation at SIGGRAPH '80 Panel on Trends on High Performance Graphic Systems, (1980).

Fuchs, H. and J. Poulton, "PIXEL-PLANES: A VLSI-Oriented Design for a Raster Graphics Engine", *VLSI Design*, Vol. II, No. 3, 3rd. Quarter, (1981).

Fuchs, H. and B. W. Johnson, "An Expandable Multiprocessor Architecture for Video Graphics", *Proceedings of 6th Annual (ACM-IEEE) Symposium on Computer Architecture*, (April 1979) 58-67.

Fuchs, H., J. Poulton, A. Paeth and A. Bell, "Developing Pixel-Planes, A Smart Memory-Based Raster Graphics System", *Proc. MIT Conference On Advanced Research*

in VLSI, Artech House, Dedham, MA., (January 1982).

Gupta, S., R. Sproull, I. E. Sutherland, "A VLSI Architecture for Updating Raster-Scan Displays", *Computer Graphics (SIGGRAPH '81 Proceedings),* Vol. 15, No. 3, (August 1981) 71-78.

Ikedo, T., "High-Speed Techniques for a 3-D Color Graphics Terminal", *IEEE Computer Graphics and Applications,* Vol. 4, No. 5 (1984).

Kay, A., "Microelectronics and the Personal Computer", *Scientific American,* Vol. 237 No. 3, (September, 1977).

Kedem, G. and J. Ellis, "Computer Structures for Curve-Solid Classification in Geometric Modelling", Technical Report TR137, Department of Computer Science, University of Rochester, (May, 1984).

Pike, R., Presentation at University of North Carolina at Chapel Hill, (1984).

Pinkham, R., M. Novak and K. Guttag, "Video RAM Excels At Fast Graphics", *Electronic Design,* (July 21, 1983) 161-172.

Schachter, B., "Computer Image Generation for Flight Simulation", *IEEE Computer Graphics and Applications,* Vol. 1, No. 4, (1981).

Schumacker, R. A., B. Brand, M. Gilland, W. Sharp, "Study for Applying Computer-generated Images to Visual Simulation", *U. S. Air Force Human Resources Lab. Tech. Rep.* AFHRL-TR-69-14, (September 1969).

Sutherland, I., "The Ultimate Display", *Proceedings of the IFIP Congress,* Vol. 2, 1965.

Thacker, C. P., E. M. McCreight, B. W. Lampson, R. F. Sproull, D. R. Boggs, "ALTO: A Personal Computer", Xerox Corp., (1979) in Siewiorek, D. P., C. G. Bell, and A. Newell, *Computer Structures: Principles and Examples,* McGraw-Hill, (1982) 549-572.

Weinberg, R., "Parallel Processing Image Synthesis and Anti-Aliasing", *Computer Graphics (SIGGRAPH '81 Proceedings),* Vol. 15, No. 3, (August 1981) 55-61.

Whelan, D., "A Rectangular Area Filling Display System Architecture", *Computer Graphics, (SIGGRAPH '82 Proceedings),* Vol. 17, No. 3, (July 1982) 147-153.

Williamson R. and P. Rickert, "Dedicated Processor Shrinks Graphics Systems to Three Chips", *Electronic Design,* (August 4, 1983) 143-148.