Distributed Processing in a High-Performance Smart Image Memory

James H. Clark and **Marc R. Hannah**, Stanford University

Most graphics systems being produced today employ an *image memory* in which one or more bits of memory are associated with each pixel on a raster-scan CRT. These memories, which are usually called "frame buffers" or "bit-mapped displays," are popular for several reasons. They are economical because raster-scan CRT technology, i.e., television, is well developed and because memory costs are low. Also, because in an image memory the *update* rate is not linked to the *refresh* rate, an arbitrarily complex picture may be deposited in the memory in a sufficiently long period of time without affecting the fixed refresh rate of the display. Finally, a high-resolution, raster-based display allows very high-quality characters to be generated at a rate that does not depend upon the complexity of the font (Knuth, 1979).

The main shortcoming of image memories is performance. Typical random update rates are about 1/2 microsecond per pixel. Thus, random operations that change the entire image might take up to 1/2 second for a 1000x1000 display and 1/8 second for 500x500 displays.

High-performance applications for which this update rate is inadequate have been unable to benefit from the low cost of these systems. For example, high-performance line, or vector, drawing applications use random-deflection CRT's because of their faster line-drawing rate, such as that of the Picture System by Evans and Sutherland Computer Corporation. These systems typically require costly power supplies to deflect the electron beam rapidly. They also require special character-generation hardware to generate characters efficiently, and they cannot adequately produce shaded polygonal areas. High-performance polygon-based systems are also expensive. They require special scan-conversion hardware to generate the pixel information in scan-line order for the CRT. Moreover, they typically have an upper limit on the number of polygon edges that can be scan-converted on a given scan-line, and their expense usually precludes their use for character display.

The performance limitations of image memories are in the time required to do the arithmetic of scan-conversion, i.e., determining which pixels to alter, and in the memory bandwidth. Normal organizations have one I/O port to the host processor and one output port to the CRT refresh controller, as shown in Figure 1. The host is often a general-purpose processor with the image memory part of its normal memory address space. Thus, scan-conversion is done by the host. The process is usually limited by the computation rate of the processor in combination with the bandwidth of the memory.

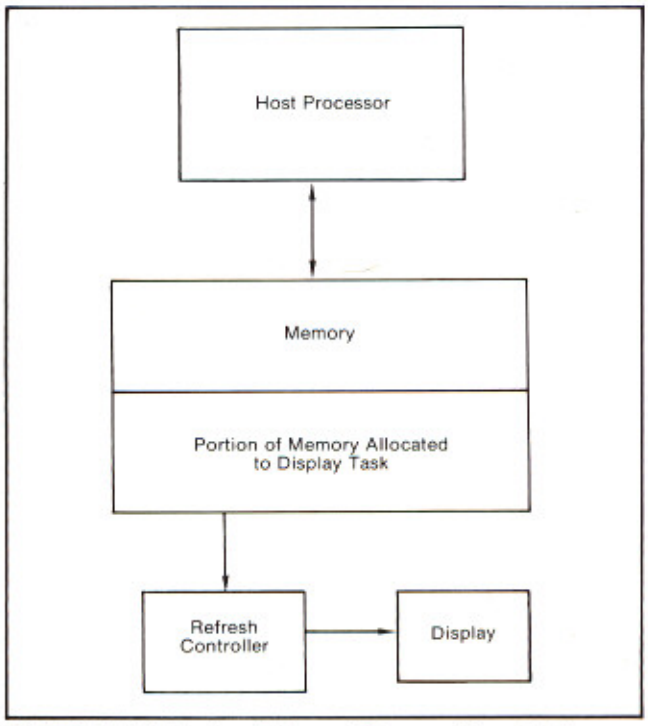

FIGURE 1. A "standard" Image Memory Organization in which the image memory is in the processor's address space, and update accesses are via the standard I/O port.

Higher-performance variations use a dedicated, microprogrammed processor for scan-conversion, and they typically interleave the memory chips horizontally to facilitate parallel access for output to the refresh controller and for fast rendering of horizontal segments. In these organizations, however, randomly-oriented segments do not take advantage of the multiple ports available in the collection of memory chips, and a single processor is used for scan-conversion.

Because image memories are usually constructed from nMOS RAM chips, the update rate is ultimately governed by the cycle time of these chips. For the most inexpensive 16k

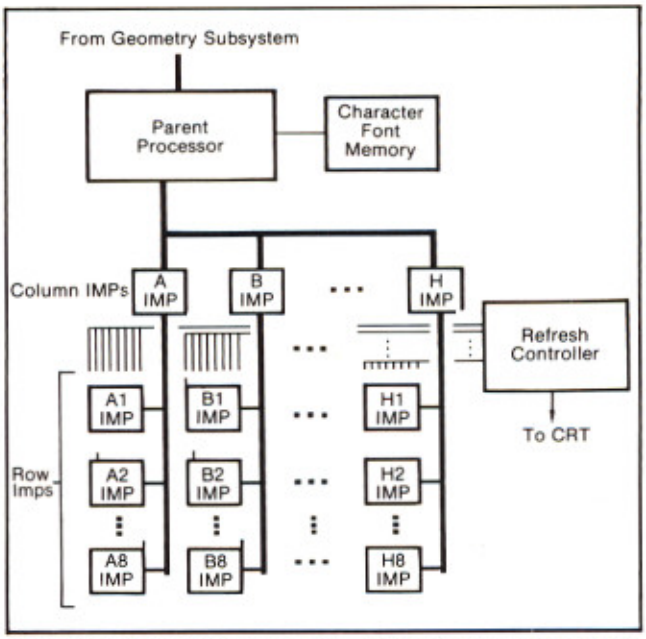

FIGURE 2. Image Memory Organization for a 1024x1024 resolution image. Each Row Processor IMP Block is actually an Image Memory Processor and one or more 16k memory chips.

RAMs, this time is about 400 nanoseconds. Normal organizations with a single update port are therefore usually bandwidth limited by this cycle time.

Any interesting image memory will have more than one of these RAM chips. Because each chip in the memory effectively has a 400-nanosecond I/O port, the maximum bandwidth of the composite memory is this time divided by the number of memory chips. For example, in a bit-mapped image with one bit per pixel, a 512×512 resolution memory requires 16 chips, and a 1024×1024 organization requires 64 chips. Thus, in a suitable organization, the maximum bandwidth attainable is either 400/16 or 400/64 nanoseconds, respectively.

This paper describes a high-performance organization that will, under the best circumstances, scan-convert a suitable set of geometric primitives and access the image memory at the maximum rate attainable. The image memory system described here scan-converts polygons, lines and characters using a custom, nMOS Image Memory Processor (IMP), which has been designed, simulated, laid out and is presently being fabricated by the Stanford University Integrated Circuits Laboratory. Multiple copies of the IMP in combination with either 16k or 64k memory chips will yield a low-cost Smart Image Memory System that should provide three improvements over existing systems: 1) vector-drawing rates comparable to those of random-deflection CRTs, 2) previously unattainable, arbitrary-font character drawing rates, and 3) polygon drawing rates comparable to those of very expensive, dedicated polygon systems. This Image System combined with the Geometry Engine System described in a previous issue of LAMBDA (Clark, 2nd Qtr., 1980) should result in a very powerful personal-computer graphics system.

System Organization

The Image Memory System for a 1024×1024 organization using 16k RAM chips is shown in Figure 2. The particular size of the memory chips and resolution shown in this figure are independent of the memory system organization; this choice is to illustrate the organization. The system uses a two-level, hierarchical busing structure with interleaved processors along each bus. All data passes through the Parent Processor on its way to the memory. This processor is not a custom chip but is a microprogrammed processor which can access the character font memory and send information asynchronously to the 8 Column IMPs (C-IMPs). Its primary purpose is to prepare the geometric primitives for scan conversion.

Each of the C-IMPs receives data from the Parent Processor and controls and supplies data to its 8 Row IMPs (R-IMPs). The IMP therefore really does more than one type of operation, but since the differences are only in the microcode and a few other minor areas and since many of the functions share microcode routines, one chip type does both functions, in a manner similar to that of the Geometry Engine organization.

Each R-IMP accepts data from its C-IMP and communicates both with the RAM chip(s) connected to it and with the refresh controller. Each R-IMP controls n memory chips, where n is the number of bits per pixel in the image memory; the current design limits this number to 10 due to pin count on the package housing the IMP. All accesses to the image memory go through the appropriate R-IMP for the pixel being accessed.

Each IMP is really two completely self-timed, asynchronous processors. One processor does the forward difference equations of scan conversion as either a C-IMP or an R-IMP, and the other processor generates the timing signals, data-refresh signals and addresses for the memory chips.

The two-level busing structure shown in Figure 2 provides a convenient interleaving of the memory chips. This interleaved structure maps onto the screen in such a way that, for any contiguous 8×8 set of pixels on the screen, each pixel is controlled by one processor and each processor controls one pixel. For example, Figure 3 shows a randomly chosen 8×8 region with the pixels labeled according to the R-IMP in charge of them. The letter designates the parent C-IMP and the number designates the R-IMP within a column. This interleaving and busing structure enables very high-performance rendering of polygons, lines and characters, as described below.

The Refresh Controller is responsible for supplying addresses to the Memory Interface Processors of the R-IMPs, both to obtain data for output to the display and to refresh the dynamic RAMs of the image memory. Except for generating memory refresh addresses, this refresh controller is like any used in conventional frame-buffer/image-memory designs. That is, it might have color lookup tables, cursor display and other similar features. By requiring that the addresses be generated externally, the IMP design is independent of the resolution of the output device, and it may be used with any scan-line refresh strategy, for example, either 30 or 60 Hertz, interlaced or non-interlaced.

IMP Organization

As mentioned above, the IMP is really two processors: a Linear Difference Engine for doing scan-conversion arithmetic and a Memory Interface Processor for generating

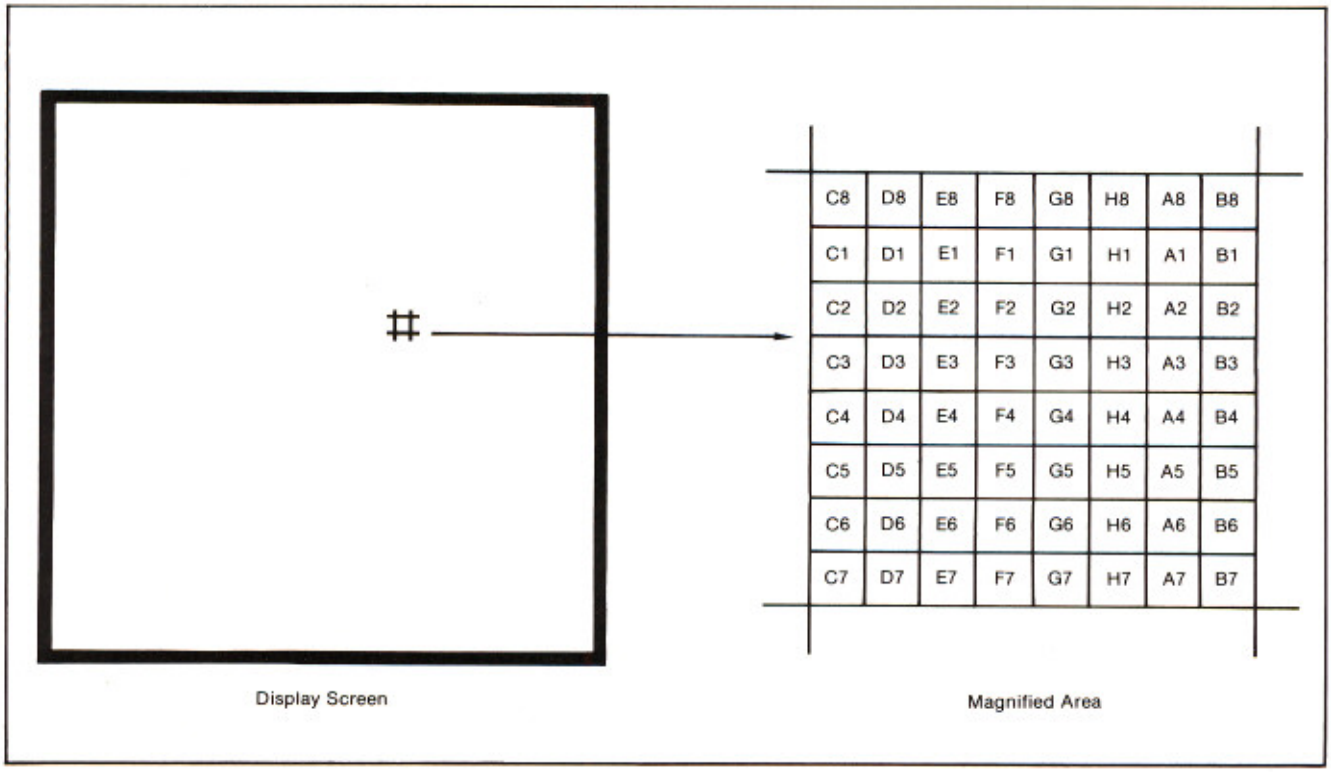

FIGURE 3. A randomly chosen 8x8 region of contiguous pixels labeled according to the R-IMP in charge of them. The letter designates the relevant C-IMP, and the number designates the R-IMP.

timing and control signals and interacting with the memory chips. Figure 4 shows a plan view of the IMP chip with the different portions labeled according to function. The layout of the chip is shown in Figure 5.

The Linear Difference Engine is a PLA-controlled processor for iterating linear difference equations and other miscellaneous functions. It is a self-timed element that communicates with its controlling processor using a double-rail protocol. It consists of two function units with a bit-slice organization of the type used in the Geometry Engine, PLA control, a self-timed clock, and microprogram counter and stack. Its microprogrammed function units are copies of the Geometry Engine function units, which were fabricated, tested and measured for timing characteristics as part of MPC79 (Conway et al., 2nd Qtr., 1980).

The Memory Interface Processor is a small, PLA-controlled processor with registers for implementing the special functions mentioned below. It is also a self-timed element with its own clock. It accesses memory either to store results given to it by the Difference Engine or to fetch the locations of memory requested by the off-chip refresh controller. Because it receives asynchronous requests from both the Difference Engine and the Refresh Controller, each of which runs with its own clock, a Seitz-Arbiter (Seitz, 1st Qtr., 1980) is used to resolve synchronization conflicts. In this way the memory chip may be cycled as frequently as needed either for updates from the Difference Engine or for output to the Refresh Controller, and the refresh strategy used is independent of the IMP design and timing.

Because of the way the image is "covered" by the R-IMPs and their memories, in the organization shown, each R-IMP receives a request for output to the Refresh Controller every

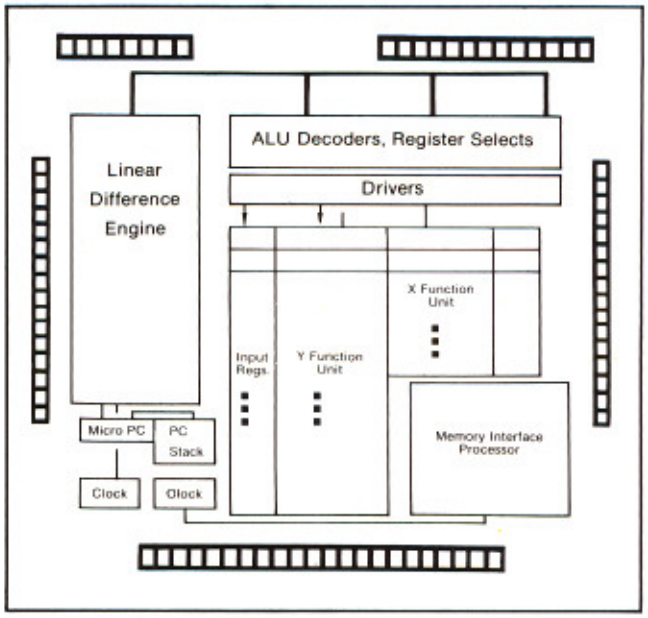

FIGURE 4. Image Memory Processor (IMP) scan-converts polygons, lines and characters into the Image Memory.

64p time units, where p is the time unit per pixel. The most demanding situation occurs if the image is refreshed at 60Hz, non-interlaced, in which case p is about 16 nanoseconds and 64p is about 1 microsecond. Thus, each microsecond, the Memory Interface will receive an output request from the Refresh Controller. Because the Memory Interface in the worst case might have just started an update sequence for the Difference Engine, the memory access sequence for the

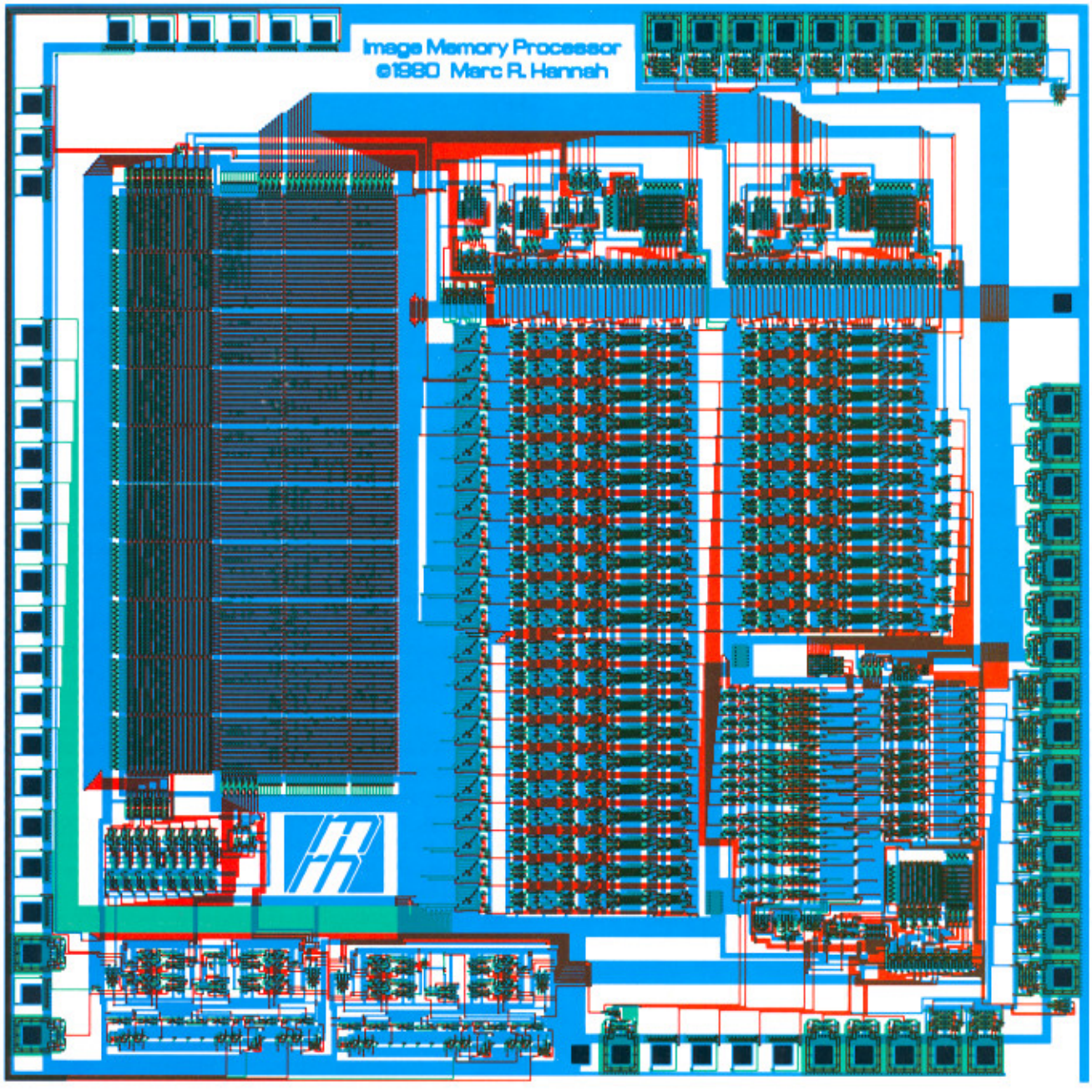

FIGURE 5. Layout of the Image Memory Processor (IMP) chip. Photo courtesy of Jim Clark and NASA/Ames, RKD Division.

Refresh Controller will begin no more than one memory access time later. Thus, one layer of buffering of the output to the Refresh Controller eliminates all problems except those resulting from synchronization conflicts. When two requests arrive "simultaneously," the Seitz-Arbiter will ensure that the system does not fail. Because the display task cannot be stopped, it is still remotely possible that the image will display a sequence of 8 pixels incorrectly on one frame once every four or five days.

A C-IMP uses only the Linear Difference Engine portion of an IMP. It receives data from the Parent Processor, does the functions described below, and sends the results to one or more of its R-IMPs in pipeline fashion. All communication is asynchronous. Because an R-IMP is connected to memory chips, it uses both the Linear Difference Engine and the Memory Interface Processor. The Difference Engine on an R-IMP receives data from its controlling C-IMP and sends results to the Memory Interface.

System and IMP Function

Data is supplied to the Parent Processor in the form of font selection commands, character codes, 2-D coordinates of points representing either endpoints of lines or vertices of polygons, and miscellaneous other commands. We will describe the characteristics of the IMP and Image System with regard to the operations carried out on lines, polygons and characters.

Line Rendering

Lines are supplied to the Parent Processor by Move and Draw commands followed by their 2-D endpoints. The endpoints are expressed in the fixed-point coordinate system of the particular organization, e.g., $0 \leq x, y \leq 1023$. The Parent Processor computes the forward difference equation for the line and sends the starting column, slope, line width and ending column to the Column IMPs (C-IMPs). Each C-IMP determines the portion of the line intersecting the column and sends that information to the appropriate R-IMPs. After the R-IMP acknowledges receipt of the information, the C-IMP proceeds to compute the intersection region of the line with the next column of the image under its control, as shown in Figure 6. Meanwhile, the R-IMP accesses the memory chip(s) under its control with the appropriate address.

For 1-bit-per-pixel organizations (binary images), the memory can be complemented, set, cleared, etc., or rendered with a stipple pattern as described below. If more than one bit per pixel is available (grey-scale image), the pixels intersected by the line can be turned on with a particular color or shade. The particular value given to the pixel is determined by a previously set shade/color register that is also part of the memory interface processor.

Because lines can have width and can be scan-converted faster than polygons, in certain applications such as IC layout (in which most polygons are either horizontal or vertical), the drawings can be generated much more rapidly using lines.

Character Rendering

In character mode, character codes and bounding box information (clipped characters) are presented to the Parent Processor, which accesses the font memory RAMs and sends the columns of the font description to the appropriate C-IMP processors. After each column code is passed with acknowledgment to the appropriate C-IMP, the Parent Processor is free to receive the next character and look up its code in the font memory. Meanwhile, each C-IMP asynchronously accepts and forwards the column code to its Row IMPs, after which it is free to receive the next column code or other drawing command. Depending upon the position of the character relative to the column, each R-IMP modifies the appropriate memory location. The shade/color register determines the value to be used in writing the character into the memory.

Polygon Rendering

In polygon mode, the Parent Processor receives the polygon command followed by a sequence of 2-D coordinates representing the vertices of the polygon. Figure 7 illustrates the parallel scan-conversion process for a three-sided polygon. The polygon must be convex, and its edges are sorted according to their leftmost vertex by the Parent Processor. Thus, the number of edges per polygon is limited by the amount of temporary local storage available to the Parent Processor.

C-IMPs accept the edges two at a time for scan-conversion. The edges are scan-converted just as with lines. For each successive column under the control of a given C-IMP, the C-IMP determines from the two active edges of the polygon a vertical span to be covered by its R-IMPs and sends them the information. After the vertical span information is taken, the C-IMP determines the vertical span for the next column controlled by it. If one of the edges of the polygon is expired, it waits for the next edge of the polygon to be sent to it by the Parent Processor.

Meanwhile, the R-IMPs compute where the vertical spans of the polygon are intersected by rows of the picture, in much the same way that the C-IMPs compute where columns intersect the edges of the polygon. After determining the pixels yo be altered, the R-IMPs make the appropriate request to the Memory Interface Processor, which alters the memory contents accordingly.

In binary images, the pure polygon mode can at most complement, set or clear the pixels covered by the polygon. Therefore, to allow different polygons to be distinguished in a binary image, an 8×8 stipple register is distributed in the Memory Interface Processors of the R-IMPs. On a binary image system, a polygon is rendered with the stipple pattern loaded into this register. The controlling processor loads the stipple register before drawing a polygon of a given type, and all subsequent polygons use this pattern as a mask in altering the memory locations. Therefore, even on a binary image, the user can draw as many different types of polygons as there are distinguishable stipple patterns. Moreover, by setting the mode register appropriately, he can cause the current polygon to overwrite all things "under" it, be OR'ed with the existing image, etc. This mode allows low-cost, high-performance renderings of integrated circuit layouts.

Image Mode

Because it is sometimes useful to be able to store an unstructured image in the memory, image mode allows a stream of pixel values to be written into the memory. Binary images can be written using the mode register as a mask.

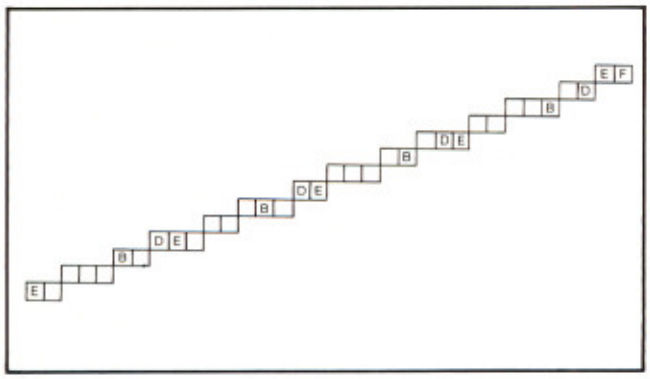

FIGURE 6. A line that begins in a column of C-IMP "E" and ends in that of C-IMP "F". All pixels labeled "E" are modified by C-IMP "E"; similarly for the other C-IMPs.

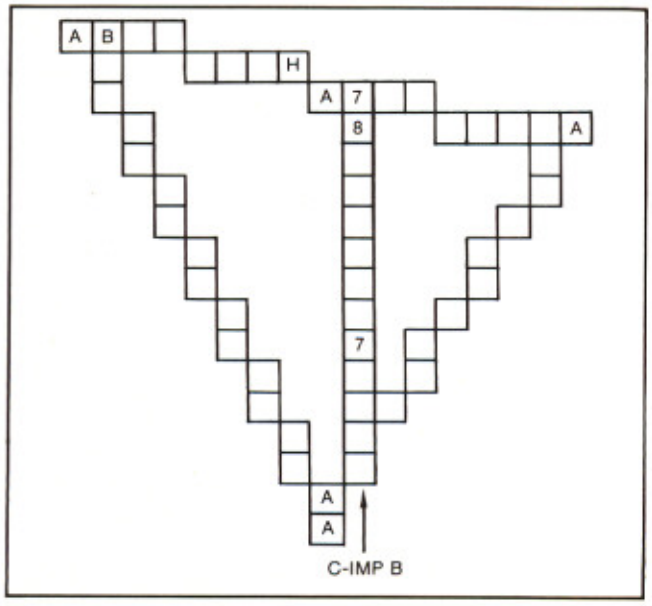

FIGURE 7. The C-IMPs scan-convert the edges of the polygon, while the R-IMPs scan-convert the interior. R-IMP B7's action is shown.

Summary

The system described here is expected to solve the problem of fast scan-conversion into an image memory. Moreover, it can be extended to a very cost-effective solution of the hidden-surface problem by using a z-buffer (Catmull, 1974).

The IMP design is an example of the type of system design that can result from understanding the limitations and power of custom nMOS design. Just as with the Geometry Engine, implementing this architecture with either standard microprocessors or standard catalogue parts would either cost too much or decrease the performance. It makes sense only because we were able to structure the primitives to be put on silicon according to the needs of the problem. Thanks to Mead and Conway and their excellent book (Mead and Conway, 1980), system architects can now approach system design in a much more innovative way.

Through the design of the IMP and the Geometry Engine, we have developed a design paradigm for microprogrammed computing engines. These projects, which were partly motivated by the need for better visual design aids, have been useful for debugging various design tools and in buiding an environment that advances our understanding of the art of designing large-scale systems on silicon. For example, one outcome of this project has been a much better understanding of the need for PLA minimization. Our micro-architecture paradigm has led to PLAs that shrink only about 5% when heuristic PLA minimization techniques are applied to them (Clark and Hennessy, in preparation). Consequently, we believe that PLA minimization really pays off only for small PLAs.

Acknowledgments

We thank Carver Mead, Lynn Conway (and Chuck Seitz) for the excellent book on VLSI design, Professor Jim Meindl of the Stanford Integrated Circuits Laboratory for his enthusiasm in fabricating the project, Professor Forest Baskett for his enthusiastic support and for organizing the computing environment, and Professor John Hennessy for providing the essential microprogram simulator.

This research was supported in part by the Defense Advanced Research Projects Agency, contract number MDA903-79-C-0680.

References

Catmull, E. 1974. "A Subdivision Algorithm for Computer Display of Curved Surfaces." University of Utah Technical Report UTech-CSL-74-133.

Clark, J.H. Second Quarter, 1980. "Structuring a VLSI System Architecture," *LAMBDA*, Vol. I, No. 2.

Clark, J.H., Hennessy, J.L. Hannah, M.R., et al., "VLSI System Control Structures," Stanford CSL Technical Report (in preparation).

Conway, L., Bell, A.G. and Newell, M.E. Second Quarter, 1980. "MPC79: A Large Scale Demonstration of a New Way to Create Systems in Silicon," *LAMBDA*, Vol. I, No. 2.

Knuth, Donald E. 1979. *TEX and METAFONT*, Digital Press.

Mead, C.A. and Conway, L.A. 1980. *Introduction to VLSI Systems*, Reading: Addison-Wesley.

Seitz, C.L. First Quarter 1980. "Ideas About Arbiters," *LAMBDA*, Vol. I, No. 1.

λ

About the Authors

James H. Clark received B.S. (1970) and M.S. (1971) degrees in Physics from Louisiana State University and a Ph.D. (1974) in Computer Science from the University of Utah. He was an Assistant Professor of Information Sciences at the University of California at Santa Cruz from 1974 to 1977 and a consultant from 1977 to 1979. In 1979 he joined the Computer Science Laboratory of the Department of Electrical Engineering of Stanford University as an Associate Professor.

His research interests are the design of computer systems, computer graphics, LSI design aids and VLSI systems architecture.

Marc Hannah is a graduate student in the Computer Systems Laboratory at Stanford University. His interests include real-time computer graphics hardware and multiprocessor systems. He received a BS from Illinois Institute of Technology in 1977 and an MS from Stanford in 1978.