PIXEL-PLANES: A VLSI-Oriented Design for a Raster Graphics Engine

Henry Fuchs and John Poulton, University of North Carolina

We present here a VLSI-oriented design for a special-purpose graphics engine capable of rapidly rendering shaded three-dimensional images from a polygonal data base onto a raster-scan color video display. The system achieves its speed by performing the most time-consuming calculations with special hardware at each pixel memory cell. These calculations (for polygon definition, visibility, and color rendering) are achieved in a distributed fashion which requires only a pair of one-bit adders and a one-bit storage element at each pixel. This circuitry is combined at each pixel with the storage elements required for a frame buffer, with only slightly more silicon area than in a conventional memory design. The most burdensome parts of the image generation task—the separate calculations for the 2^{18} to 2^{20} pixels—can therefore be carried out in parallel at all the pixels in the entire image. The other, less demanding computations (coordinate transformations, clipping, perspective scaling, and lighting) can easily be handled in real time by current graphics systems. One such system is the host for our raster-graphics engine. Although we cannot give precise system execution times (because the first chips are just now being fabricated) we estimate that the system will be able to process 15,000 to 30,000 polygons per second.

The Pixel-planes system consists of two parts: 1) a special-purpose computer (the 'pre-processor') which converts polygon data from the host into a form suitable for transmission to 2) a set of identical 'smart' memory chips. The system performs visibility calculations using the depth-buffer (z-buffer) algorithm, and can also execute a Gouraud-like smooth shading algorithm. Important features of the system are:

- Visibility and painting calculations are performed polygon-by-polygon rather than in scanline order.
- The expected polygon processing time during image generation is as fast as line-processing in current real-time line-drawing systems.
- Processing time for a polygon is independent of the size and orientation of that polygon, and increases only linearly with the number of vertices. Convex polygons with any number of vertices can be processed.
- The 'smart' frame-buffer memory, implemented in a number of identical chips, is easily expandable to accommodate increased screen resolution.

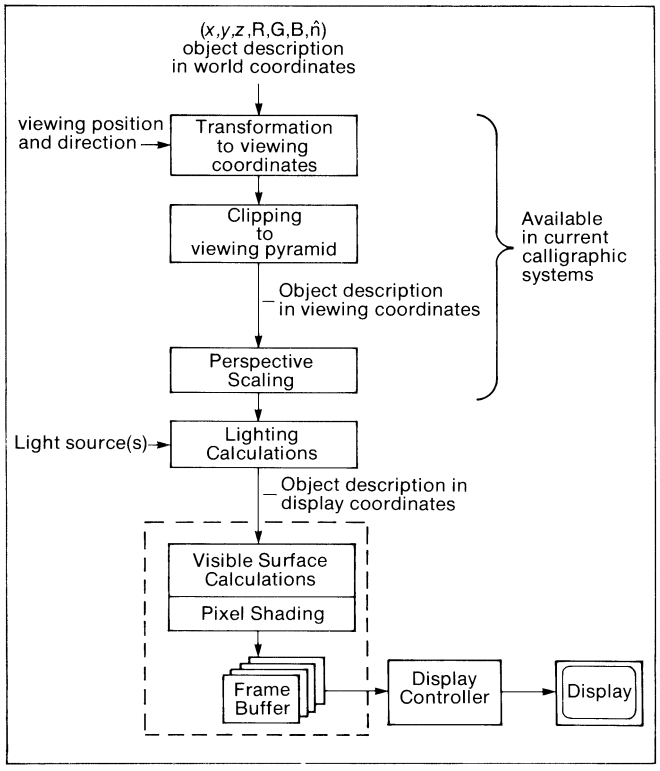


FIGURE 1: Steps required in raster-scan rendering of solid objects. Dashed outline indicates scope of new Pixel-planes hardware.

- The design retains the regularity and simplicity of conventional memories, to take advantage of well-developed techniques for memory system design; thus, layout, testing, and communication are relatively simple.
- The system is double-buffered to enable smooth transitions between successive images.
- Line and two-dimensional objects can also be handled by the system. Such objects can be processed somewhat more rapidly than smooth-shaded solid objects.

Raster-Scan Graphics Fundamentals

There are many ways of rendering a raster-scan image from a polygonal data base (Sutherland, Sproull, and Schumaker, 1974). One such method is outlined in Figure 1 (Newman and Sproull, 1979). The data base contains a de-

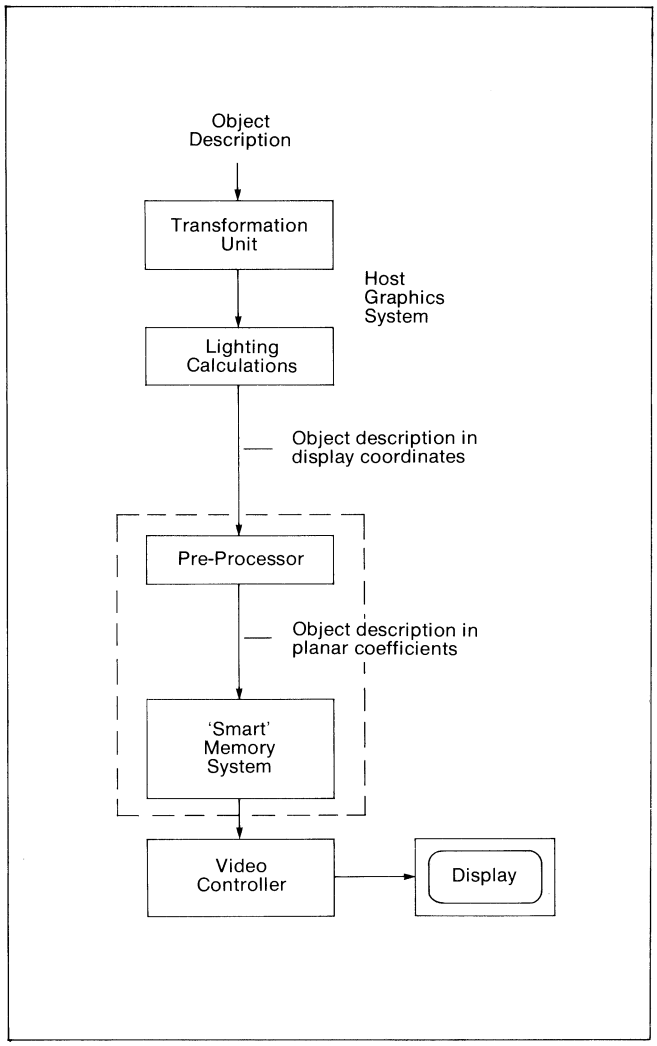


FIGURE 2: Overall system organization. Dashed line shows Pixel-planes hardware.

scription of a scene containing one or more objects, each of which is described by a set of (planar) polygons which approximate its surface. (The current version of our system lets polygons have any number of sides, but restricts the polygons to convex shapes.) Polygons are processed by the graphics system one at a time, in any order. Each polygon is described by a sequence of vertices whose x,y,z coordinates are expressed in the 'world' coordinate system. Associated with each vertex is the triple R,G,B specifying the intrinsic vertex colors, and components of the unit normal to the surface at the vertex. When a polygon is processed, the coordinates of the vertices are first transformed to viewing coordinates according to the viewing position and direction. The polygon is then clipped to the viewing pyramid, eliminating portions of the polygon outside the field of view. Next the transformed and clipped polygon (now in viewing coordinates) is scaled for an appearance of perspective and reexpressed in the coordinates of the display device. Then, lighting calculations are performed in which the precise color at each vertex is calculated based on the direction and distance to the light source(s), the intrinsic vertex color, surface reflectivity, and other factors, if desired. These calculations result in a new R,G,B triple for each vertex, representing the light reflected by the object toward the viewer.

Up to this point, the number of calculations in each step depends only on the number of polygon vertices in a scene. The above geometric transformations are carried out in real time by current calligraphic systems, such as the Vector General 3303 or the Evans & Sutherland Picture System II. A recent VLSI-oriented project (Clark, 1980) promises to make such transformation engines even more efficient and less expensive. The much more burdensome subsequent calcuations require the graphics system to:

a) identify all pixels which lie within the current polygon,
b) determine the visible pixels of each polygon, and
c) paint each such pixel.

Pixel-planes performs the above three sub-tasks simultaneously for all pixels in the entire frame buffer. Therefore, it may be connected to a transformation unit (in, say, a line-drawing system) to form a complete system for rapid rendering of shaded polygonal images. The memory portion of the system serves as a frame buffer from which a controller can refresh a video display.

System Description

Figure 2 is a diagram of the overall system. The host graphics system contains a transformation unit which carries out viewpoint transformations, clipping, and perspective scaling, as outlined above. The host also performs lighting calculations at each vertex. Data output by the host is passed to the Pixel-planes pre-processor, a transformation unit which converts polygon vertex data into coefficients of planar equations (the form required by the enhanced memory system). The fundamental operation of the memory system is the calculation, simultaneously at each pixel, of the function $F(x,y)=Ax+By+C$, where x and y are the coordinates of the pixel in the display space. The process of painting a polygon involves:

a) Processing polygon edges in sequence, where each edge is encoded in the coefficients A,B,C of a linear equation $F(x,y)=Ax+By+C=0$.
b) Performing the depth-buffer computation in which the z-coordinate of each pixel is encoded in another set of coefficients in the planar equation $z=F(x,y)$.
c) Carrying out a shading algorithm in which the intensity surface for each color in the polygon is approximated by a series of planes each encoded in the coefficients of three planar equations $R,G,B=F(x,y)$.

Figure 3 is a (conceptual) diagram of the memory system. The system consists of an array of identical memory cells connected to a grid which carries data to the cells. Two serial multiplier trees, which we designed, appear at the top and left-hand side of the array. These multipliers accept data from the pre-processor and calculate, simultaneously for all values of x or y, values for the functions $Ax+C'$ and $By+C''$. A bit-serial representation of these functions is placed on the memory grid. Each memory cell contains an adder which calculates the sum of these two functions in order to generate $F(x,y)=Ax+By+C'+C''$. (The separation of the constant C into $C'+C''$ is for convenience, as shown below.) A block diagram for the x-multiplier is shown in

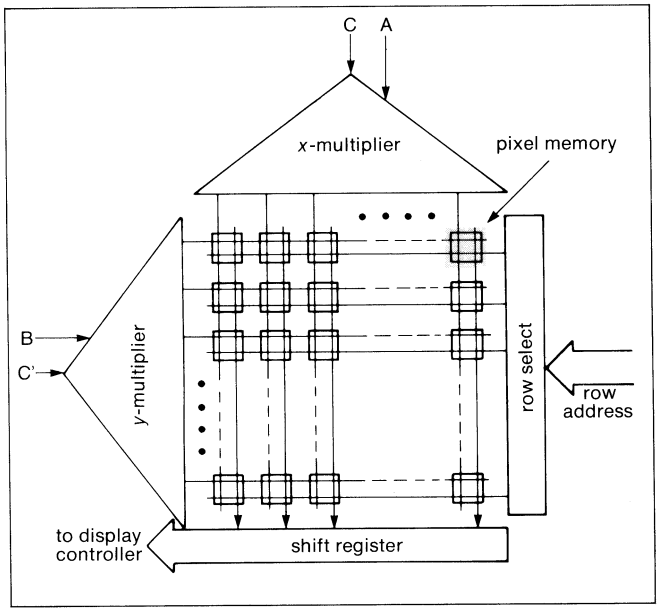


FIGURE 3: Organization of 'smart' memory system.

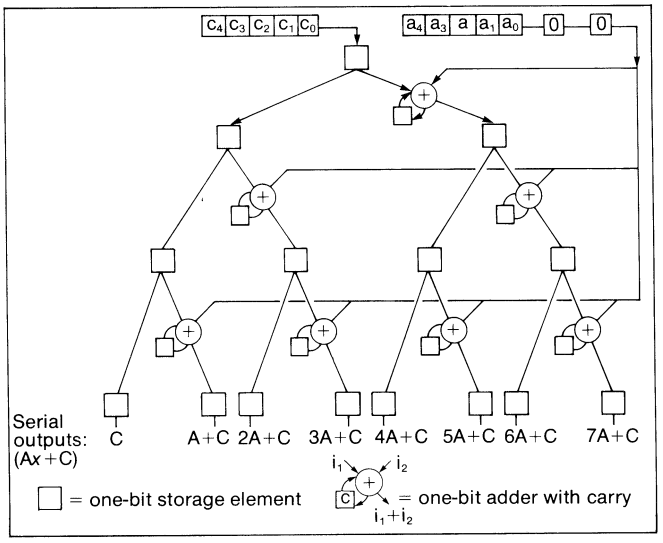


FIGURE 4: The x-multiplier tree layout.

Figure 4. (The y-multiplier is identical.) Data for the coefficients A and C' is input to the multiplier in bit-serial form. As shown, the multiplier calculates the expression $Ax+C'$ for all values of x in the range of the display (0 to 511 typical). The y-multiplier accepts bit-serial data for coefficients B and C'' and generates the expression $By+C''$ synchronously with the x-multiplier's $Ax+C'$.

Figure 3 shows a conceptual scheme for raster-scanning the memory cells. A row-select decoder driven by the display refresh controller selects a row of pixel memory cells, whose data is output serially to a shift register. This shift register then allows the video data to be shifted out to the refresh controller. Because of the very large bandwidth required for video data, this scheme would be modified in practice by transmitting data from a number of neighboring cells simultaneously over a wide parallel data path, and by multiplexing the shift register.

Figure 5 shows the structure of an individual pixel memory cell. The memory cell contains four registers: Z, which contains the smallest z-value so far received at the pixel (portion of displayed object closest, and therefore most visible, to the viewer); F, which provides temporary storage for the function ($F(x,y)$ output by the adder; I, the image register in which the results of the current polygon 'painting' operation are stored; and P, in which the intensity values for the previously constructed complete image are stored. The image stored in the P registers is the one currently being displayed. The P registers can be accessed by the display refresh circuitry independently of any processing operations in the pixel cells. Registers I and P can each store either a single intensity value (B/W) or three intensity values (R,G,B) in successive portions of the register. A 'control decoder' receives control signals broadcast synchronously with A,B,C',C'' coefficient data, monitors the adder and comparator, controls the flow of data among the four registers, and sets the state of a one-bit 'Enable' register (En) which determines at any time whether the cell is enabled for further operations.

'Smart' Memory Subsystem

1) When a new scene is to be painted, a control word broadcast to all memory cells causes the Z registers to be preset to all 1's (infinite depth); polygon processing then begins.
2) When a new polygon is to be processed, a control signal sets the one-bit Enable registers (En) in all cells, enabling them for operation. Each polygon is processed by a sequence of three operations:
   a) *Edge definition*: Data representing successive edges of the polygon are encoded in the coefficients of $F(x,y)$ and transmitted to the memory system. The coefficients A,B,C',C'' are chosen such that $F(x,y)$ is positive or zero for pixels (x,y values) which lie inside the current polygon, and negative for pixels outside it. As the data for each edge is received at a memory cell, the sign bit for F is checked at the termination of the data. If the sign bit is set, the En register in the cell is cleared and the cell is disabled until the next polygon is processed. Successive edges disable more and more cells as pixels fall outside a polygon edge. When all edges of the current polygon have been transmitted and processed in this way, only those pixels inside the polygon are enabled for further processing; all others have been turned off.
   b) *Z-buffer calculation*: The planar equation for the polygon is encoded in the form $z = F(x,y) = Ax+By+C'+C''$ and transmitted to the memory system. Each pixel cell still enabled subtracts in bit-serial fashion the z-coordinate stored in Z from the newly received z-value, using a second one-bit adder (the 'Comparator'). The current value of z is stored in F. At the end of transmission of z-data, Z and F are compared by checking the sign bit from the Comparator. If F>Z, then the portion of the polygon at the pixel's location is hidden and the En register is cleared, disabling further processing. If, however, F<Z, the pixel is visible; in this case the control decoder causes the contents of F to be loaded into Z, thereby updating the z-buffer.

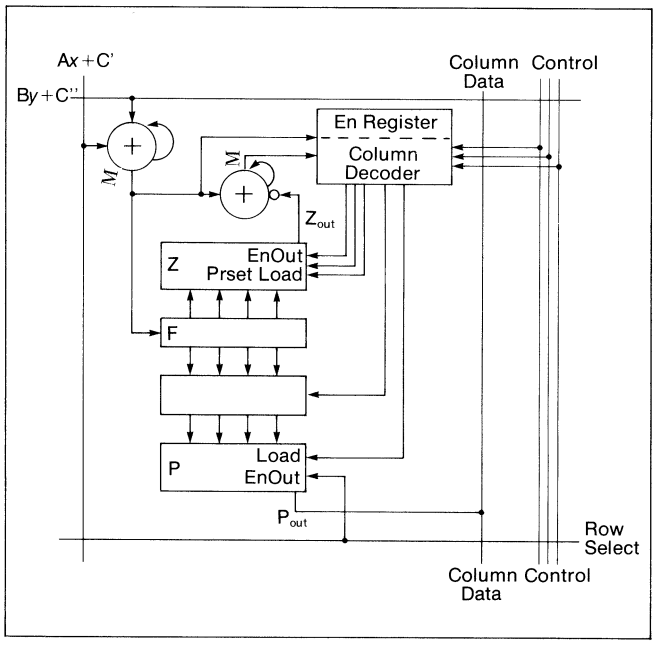


FIGURE 5: Pixel memory cell.

c) *Pixel shading*: The color intensities for the polygon are encoded in the form of planar equations I=Ax+By+C'+C'', where I is, successively, R, G, B. (This assumes that the set of values x,y,I for the polygon vertices form a plane; this is not the case in general, as discussed below.) These values for R, G, and B are stored in successive portions of register I.

3) When all polygons in a scene have been painted in this way, a control word is broadcast which causes the contents of registers I to be loaded into registers P, causing the newly completed image to appear on the display.

Pre-processor Subsystem

As shown in Figure 2, a single pre-processor unit converts the vertex data from the host into the planar-coefficient format needed by the 'smart' memory system. It is assumed that the vertices will be transmitted in order, such that each polygon is traversed in the same direction (*e.g.*, counter-clockwise), and that the last vertex will not be transmitted, because it is the same as the first. Also assumed is a control signal to indicate completion of each polygon. The pre-processor in the Pixel-planes system must convert this stream of data into the coefficients to be processed by the memory system. It must also take into account the problem of the shading algorithm used by the system. As noted above, this algorithm assumes that the shading surface for a polygon is planar (*i.e.*, that the set of coordinates x,y,I for a polygon lie in a plane). Since this is unlikely to be true for an arbitrary polygon, the pre-processer must first break an n-sided polygon into n-2 triangles, each of which shares a common vertex (the first one processed). Planar equations can then be calculated separately for each triangle. The pre-processor consists of two parts, as shown in Figure 6. An input unit stores the first vertex received for a given polygon, and re-transmits the vertex data in the order required by the coefficient unit.

The coefficients for expressions representing the edges of

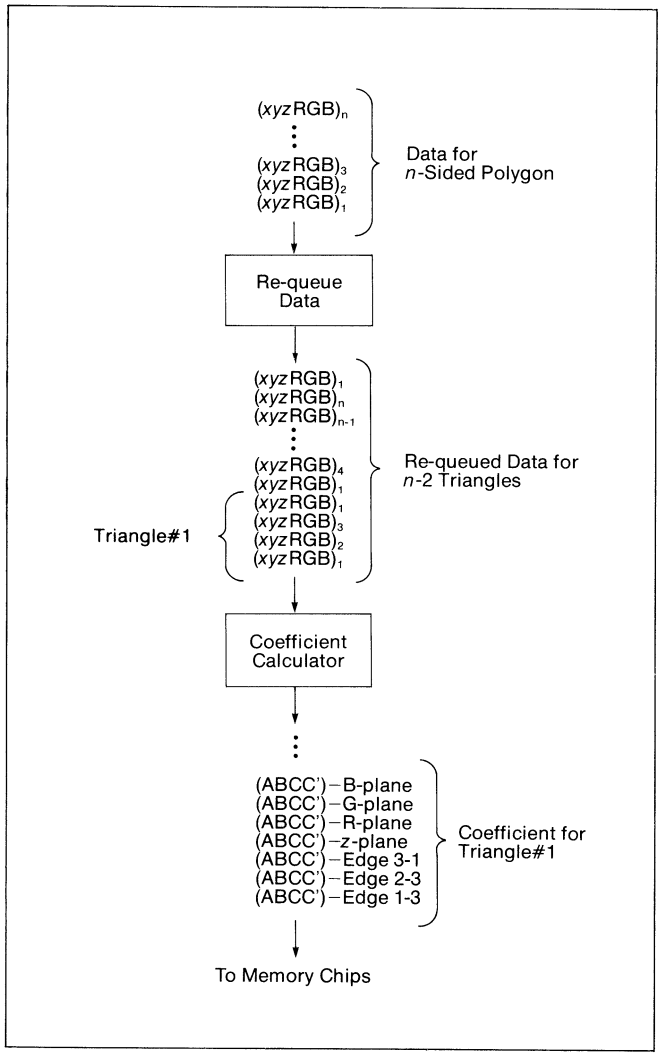


FIGURE 6: Operations in pre-processor.

a polygon can readily be calculated by taking differences in the x and y values for successive vertices. If the polygon is traversed counterclockwise, the equation for the line connecting successive vertices is:

$$-[y(i+1)-y(i)]x + [x(i+1)-x(i)]y$$
$$+[y(i+1)-y(i)]x(i) - [x(i+1)-x(i)]y(i) = 0$$

from which the coefficients A,B,C',C'' for the ith edge are

$$\begin{aligned} A &= -[y(i+1)-y(i)] \\ B &= [x(i+1)-x(i)] \\ C' &= -x(i)*A \\ C'' &= -y(i)*B \end{aligned}$$

Calculation of the planar equation coefficients in +C'' is somewhat more complicated. The equation can be found by calculating the plane which passes through a given point and is perpendicular to a given line. This line can be found by forming the vector product of the first two directed line segments in the polygon:

$$\begin{vmatrix} x & y & z \\ (x2-x1) & (y2-y1) & (z2-z1) \\ (x3-x2) & (y3-y2) & (z3-z2) \end{vmatrix}$$

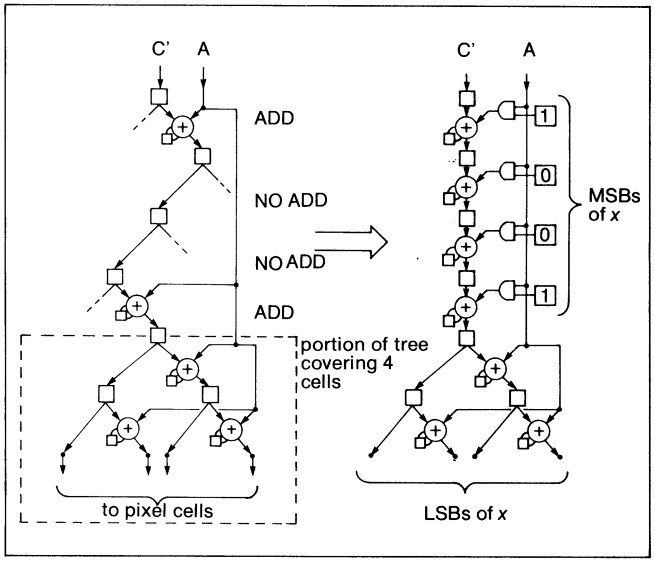


FIGURE 7: Scheme for including copy of multiplier tree on each memory chip. (The x-multiplier is shown.) ADD and NO-ADD branching decisions are shown in traversal of tree equivalent to standard serial multiplier with MSB's of x stored in x-multiplier register. (This number is also the x-address of chip.)

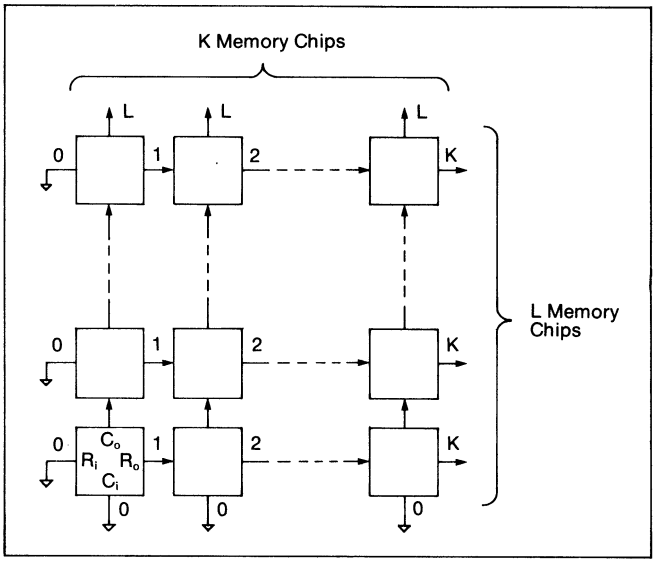


FIGURE 8: Scheme for automatically initializing chip x and y multiplier registers (and x,y addresses).

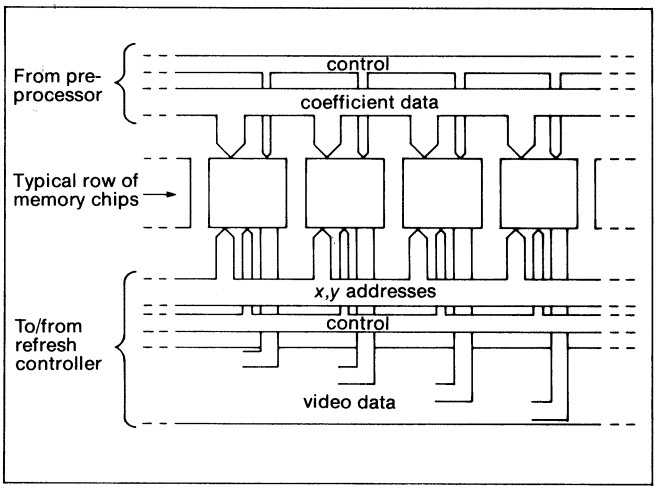


FIGURE 9: Communications with Pixel-planes memory system.

which gives a line with direction numbers

$$a = (y2-y1)(z3-z2)-(y3-y2)(z2-z1)$$
$$b = (z2-z1)(x3-x2)-(z3-z2)(x2-x1)$$
$$c = (x2-x1)(y3-y2)-(x3-x2)(y2-y1)$$

The equation of the plane which is perpendicular to this line and which passes through the point $x1, y1, z1$ is

$$z = (-a/c)x + (-b/c)y + (a/c)x1 + (b/c)y1 + z1$$

from which the coefficients

$$A = -a/c$$
$$B = -b/c$$
$$C' = (a/c)x1 = -A*x1$$
$$C'' = (b/c)y1 + z1 = -B*y1 + z1$$

are obtained. The planar equations for the red, green, and blue intensity planes are obtained in the same way by replacing $z(n)$ by R(n), G(n), and B(n) successively. The edge equation coefficients and each of the coefficients for each of the planes can be calculated using pipelined multipliers (Lyon, 1976) and serial dividers. We note several simplifications in the above formulae: In general, $C' = -A*x1$ and $C'' = -B*y1$ [$+z1$], simplifying the calculations of C' and C''. Furthermore, the direction number c from each of the vector cross-products is the same for the z-, R-, G- and B-plane calculations, and only needs to be evaluated once.

System Speed

Speed can be estimated from the time required by the memory system to complete the computation of $F(x,y)$. If x- and y-coordinates are represented as K-bit numbers, z-coordinate as an L-bit number, color intensities as M bits each, and screen resolution as N bits, then K+N+2 clock cycles would be required to calculate the edge function $F(x,y)$, L+N+2 clock cycles would be required for the z-coordinate, and 3*(M+N+2) cycles for the three color-intensity planes. Thus, for an n-sided polygon, [3K + L + 3M+7(N+2)](n−2) clock cycles would be needed to process edge, z-coordinate, and color-intensity data where smooth shading is required. Assuming, for example, K=10, L=16, M=8, N=10, and a clock period of 200 nsec, processing a four-sided polygon would require 62 microseconds. At this rate, approximately 540 such polygons could be processed in one refresh period from a 30Hz display system. Where smooth shading is not required, only a single color-intensity plane would be needed for an n-sided polygon, and processing would take only [K+N+2]n+L+3M+4(N+2) clock cycles. Under the assumptions, 940 such 4-sided polygons could be processed in one refresh period.

Implementation

The Pixel-planes systems achieves a very regular structure by including a copy of the entire multiplier tree on each memory chip. Therefore, the system can be im-

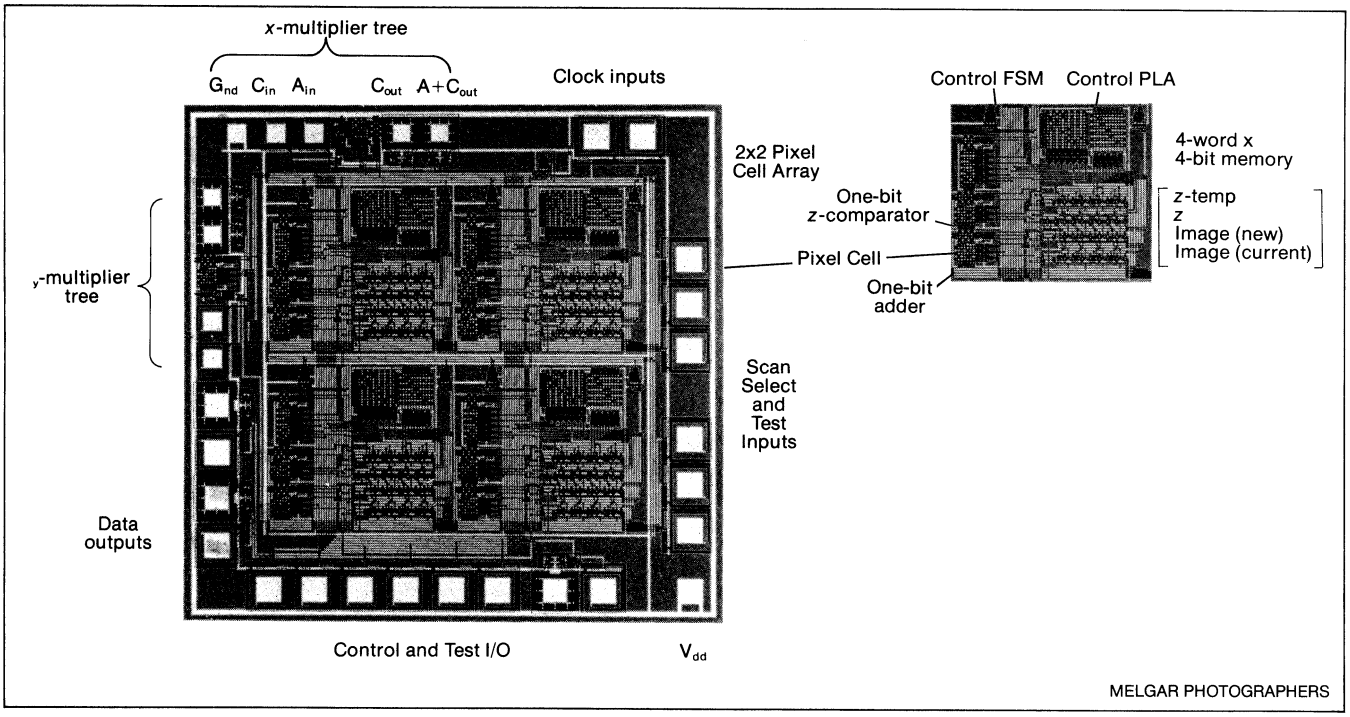


FIGURE 10: Chip layout.

plemented entirely with an array of identical chips. Little additional chip area is required for the redundant portions of the multipliers. For example, in a system designed for 512x512 resolution in which each chip contains 16x32 pixel cells, the additional hardware needed for multiplier copies would be 9 one-bit adders and 9 one-bit storage elements. The portion of the multiplier tree needed on a given chip can be determined according to Figure 7. At each branch point in the tree, the multiplier A must either be ADDED or NOT ADDED to the coefficient C'. (The coding of these ADD and NOT-ADD decisions into x- and y-multiplier registers is shown in the figure. The result is a standard serial multiplier. The multiplier registers in the serial multipliers on a given chip contain numbers which are identical to the x- and y-addresses for that chip; this fact can be used to advantage in the addressing scheme.

Figure 8 shows a scheme for setting these coefficients automatically in an array of identical chips. When the system is initialized, the chip at the lower left receives a serial stream of 0's at a column and a row input. Zeroes are loaded into the x- and y-multiplier registers in this chip; the chip then adds 1 to each of the multipliers and outputs the results serially in row and column outputs. Neighboring chips above and to the right receive these value of x- and y-multipliers, load their multiplier registers, add 1 to the multipliers, and pass them to neighboring chips above and to the right. In this way, address information passes serially through the entire memory array and eventually arrives on the upper and right-hand edges of the array, where it can be checked for correct value. A double-rail signalling convention would be used for this data transfer among chips (Mead and Conway, 1980).

The address of a given chip for the raster-scan wiring is also determined in this way. Data communications in such a system is simple: coefficient data is broadcast from the pre-processor to all memory chips simultaneously over a parallel data path of sufficient bandwidth to match the internal serial processing speed. Raster-scan addresses are likewise broadcast to all chips; each chip recognizes its own address, stored in the x- and y-multiplier registers. The x- and y-addresses can be multiplexed, because the y-address need only be changed and latched into the memory chips at the end of a horizontal scan. The x-addressing must be interleaved so that neighboring locations on a scan line of the display are stored in memory elements on separate chips. In this way, N pixels are read simultaneously from N neighboring chips onto a data bus (N pixels wide) of sufficient bandwidth to keep pace with the requirements of the display refresh (see Figure 9).

Current Work

We have designed and fabricated a very small memory chip of the kind described above; testing is now underway. This design project was undertaken as part of a VLSI systems course taught at the University of North Carolina Department of Computer Science, in fall 1980. *Introduction to VLSI Systems* (Mead and Conway, 1980) was the text, and the approach set forth therein greatly influenced our design efforts. The current chip implements one branch each of the x- and y-multiplier trees, and a 2x2 pixel memory array. This design was our first nMOS design project. We felt that a relatively small project utilizing standard cells where possible would be a useful learning experience; therefore, the chip uses space very inefficiently. (All memory cells are constructed using shift registers, and the adders and control circuits were implemented as standard-cell PLA's.) Nevertheless, the chip contains enough circuitry to form a nearly self-contained memory system, and it will be useful for testing most of the elements needed for a graphics system of practical size. In particular, each chip can be connected to

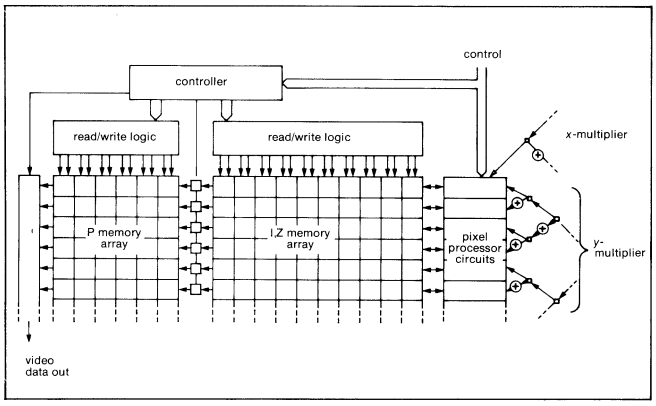


FIGURE 11: Block diagram for memory system based on dynamic RAM array. Because all pixel processors execute the same operation concurrently, read / write logic can drive all cells of memory arrays. When a new image has been constructed, any portion of the I,Z array can be loaded into the P array. The P array is accessed by address decoders for display refresh. Only the portion of memory associated with one node of the x-multiplier is shown.

others to form a larger system. Figure 10 is a photograph of the chip layout.

Future plans include designing of a much larger memory chip using dynamic RAM cells for the pixel memory. This implementation will include the ability to trade display spatial resolution for increased color-intensity resolution, and will be expandable to a display of practical size. The design of the pre-processor is also under way. The first full-scale system will use the computational and display portions of one of our in-house graphics systems.

Further Enhancements

- *No F register needed*: Referring to Figure 5 showing the registers within a pixel cell, we note that the F register is not necessary. If it is eliminated, however, the z-coefficients must be passed twice (once for comparison with the z stored at each pixel, and a second time to load the new value of z in the appropriate (visible) pixels). With this saving, the amount of memory in the system is the same as that required for a conventional frame buffer in which the z-buffer algorithm is to be executed.
- *Dynamic RAM cells*: Figure 11 shows a rough layout for the next version of the memory system, using three-transistor dynamic RAM cells (Lyon, 1980). Placing the bits for a pixel along a single column of the memory array allows 1) a flexible allocation of the bits among the Z, I, and P registers and 2) one control circuit to be shared by all RAM cells.
- *Min-Max decoders*: Additional circuitry could be added to the memory system to allow only pixels within a specified rectangular region to be enabled, this region to be specified by $x\min$, $x\max$, $y\min$, $y\max$ (Fuchs, *et al.* 1981). This circuitry would be especially useful for (colored) line drawings, which can be generated by specifying the end-points to the rectangle-enable logic and by passing only one set of planar coefficients. With this modification, some degree of anti-aliased line drawing may also be possible.
- *Concave polygons*: The 'Min-Max' decoders would also let our system handle concave polygons. (This method was suggested to us in a private communication from Satish Gupta and Mark Faust at Carnegie-Mellon University.) The technique is based on the classic notion of determining whether a point is inside or outside a concave polygon by counting the number of edge crossings which are encountered on a line between the point and a known outside point (*e.g.*, the right edge of the screen).
- *Additional multipliers*: Performance could be further improved by adding another pair of multipliers (or more). With two multipliers each for x- and y-coefficients, two planar functions could be calculated simultaneously, thereby doubling the speed of the system at the expense of relatively little silicon area.

References

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

Fuchs, H., S.F. Weiss, and J.M. Fitzpatrick. 1981. "A Graphics Memory System for Rapid Rendering of Filled Rectangles" (in preparation).

Lyon,R.F. 1980. "Signal Processing with VLSI" (notes for a constantly changing talk), Xerox Research Center, Palo Alto, Calif

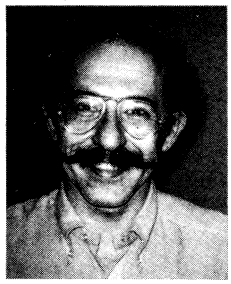
Lyon, R.F. April 1976. "Two's Complement Pipeline Multipliers," *IEEE Transactions on Communciations*, Vol. COM-24, pp. 418-425.

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

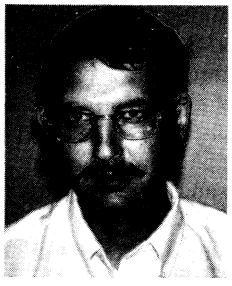
Newman, W.M. and R.F. Sproull. 1979. *Principles of Interactive Computer Graphics, Second Edition*. New York: McGraw-Hill.

Sutherland, I.E., R.F. Sproull, and R.A. Schumacker. March 1974. "A Characterization of Ten Hidden-Surface Algorithms," *Computing Survey*, 6(1):1.

About the Authors



Henry Fuchs 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. From 1975 until 1978 he was an assistant professor of Mathematical Sciences at the University of Texas at Dallas. Since 1978 he has been an Associate Professor of Computer Science at the University of North Carolina at Chapel Hill. He has participated in the founding and development of the Microelectronics Center of North Carolina, he is an independent consultant to research and industrial groups; his interests include graphic systems, VLSI architectures, computer-aided design, man-machine interaction, and medical imaging.



John W. Poulton received his BS in Physics from Virginia Polytechnic Institute in 1967, his MS in Physics from the State University of New York at Stony Brook in 1970, and PhD in Physics from the University of North Carolina at Chapel Hill in 1980. He recently joined the Microsystems Laboratory of the Department of Computer Science at the University of North Carolina at Chapel Hill. His research interests include VLSI systems design and interactive computer graphics.