DEVELOPING PIXEL-PLANES, A SMART MEMORY-BASED RASTER GRAPHICS SYSTEM

Henry Fuchs*, John Poulton*, Alan Paeth**, Alan Bell**

*Department of Computer Science, University of North Carolina at Chapel Hill
**VLSI System Design Area, Xerox Palo Alto Research Center

Abstract

We describe recent developments of Pixel-planes, a smart memory-based raster graphic system designed to support graphical interaction with 3D color rendered surfaces at a level that today is only possible with simple "wire-frame" line drawings.

In this system we replace the standard RAM chips forming the video image buffer with chips of our own design whose individual "pixels" each have a small amount of processing circuitry. Although this circuitry performs virtually all of the image-related processing, it is sufficiently simple that, in our current layout, the total silicon area is only about double that of unenhanced memories.

These chips identify the pixels that are inside each polygon, identify the subset of these pixels unobstructed by previous polygons, and smoothly color the visible pixels.

Two versions of Pixel-planes have been designed and fabricated. The first, Pixel-planes I, was a very simple version, a mere 4 pixels per chip. The second, shown in Figure 1, is a more extensive version, with a compact layout containing 64 16-bit pixels. Testing is under way.

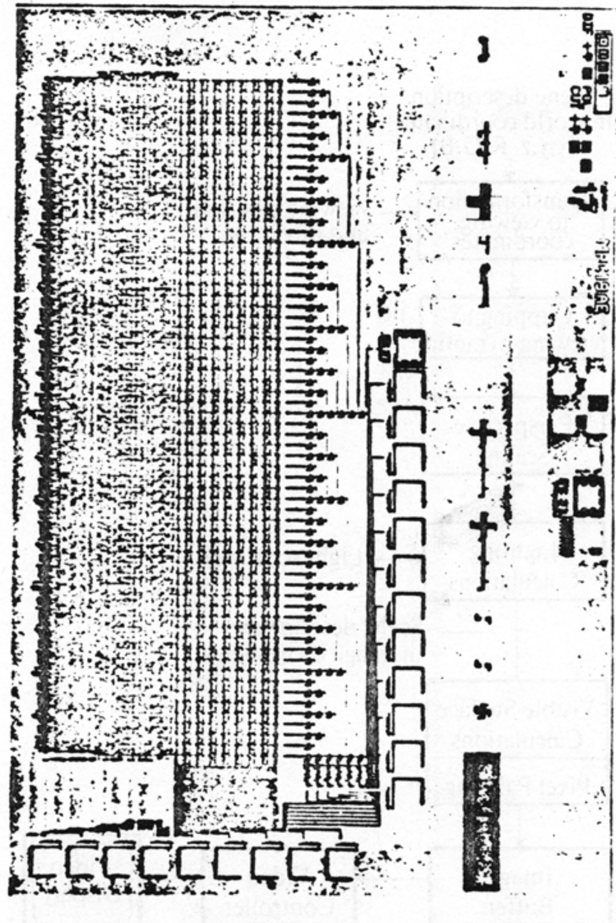


Figure 1

Photomicrograph of Pixel-planes II (Melgar Photographers).

Image Generation Overview

The rapid generation of realistic images of 3D scenes has been one of the central problems in computer graphics since the mid-1960's [Newman and Sproull, 1979]. In this section we give a brief overview of this problem.

The canonical steps for generating such an image are shown in Figure 2. The data base contains a scene description of one or more objects, each of which is described by a set of convex polygons that approximates its surface. Polygons are processed sequentially in any order. Each polygon is described by a sequence of vertices whose x,y,z coordinates are in the 'world' coordinate system. Associated with each vertex is the triple R,G,B specifying the color of the polygon at that vertex.

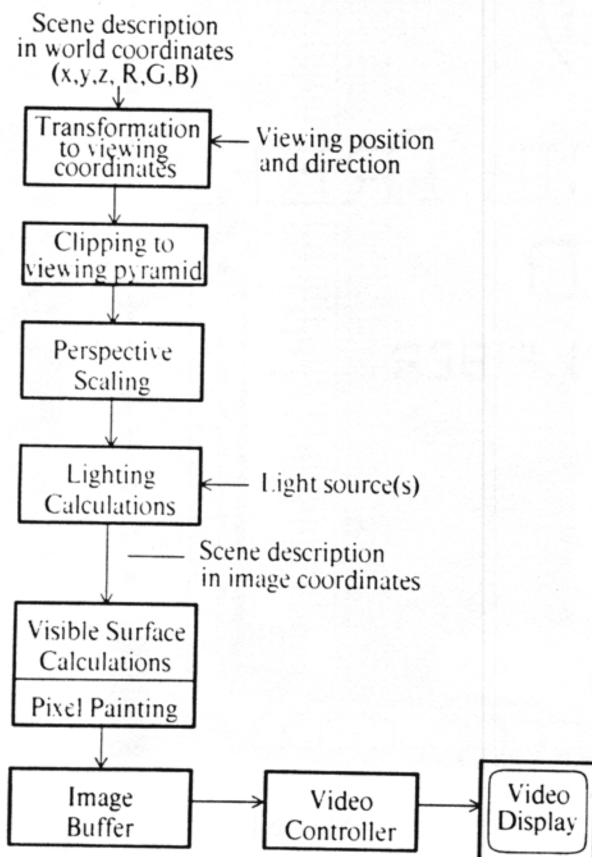


Figure 2
Digital Scene Generation

Processing of a polygon begins with the coordinates of its vertices being transformed to the coordinate system associated with the current 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 re-expressed in the coordinates of the display device.

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 (original) vertex color, surface reflectivity, and perhaps other factors. These calculations result in a new R,G,B triple for each vertex representing the light reflected by the object toward the viewer.

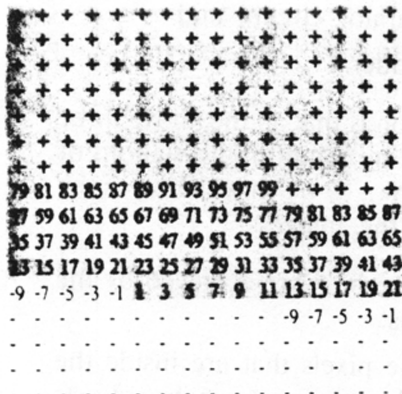
Finally the visibility calculations need to be performed. These are the most computationally intensive steps. These calculations must find the polygon that is visible at each pixel in the image. The shading and color of each pixel is then computed from the color of the vertices of the appropriate polygon.

The computational burden of visibility and pixel calculations is so formidable that real-time systems that need these capabilities, such as the digital scene generators in flight simulators, sell for $1M. The systems that don't need them, such as the vector or "calligraphic" systems, can be sold for $50K. The goal of the Pixel-planes system is to utilize the potential of VLSI technology to overcome this bottleneck [Mead and Conway, 1980].

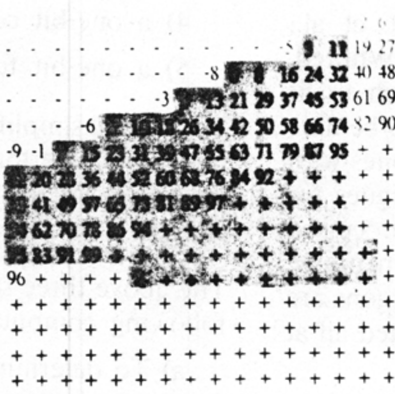
Pixel Calculation Algorithms

Pixel-planes is designed to interface to a graphic host that performs the geometic transformations, clipping, perspective, and lighting calculations on each polygon before passing it to Pixel-planes. Thus the input to Pixel-planes is a set of polygons in image space.

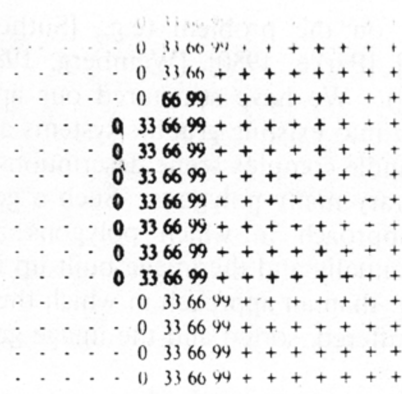
The visibility calculations can be structured in many different ways; there is a sizable literature of



a) after first edge

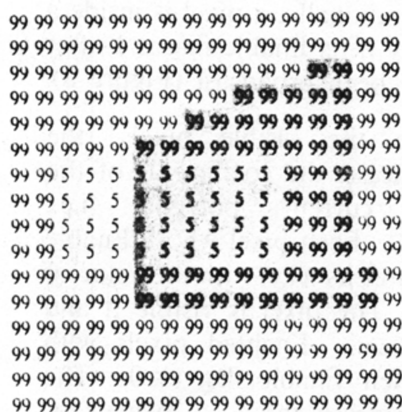


b) after third edge



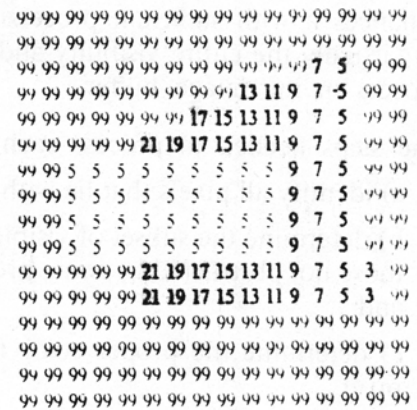
c) after final edge

Figure 3: Enable flags and f(x,y) values during early phases of processing a polygon.



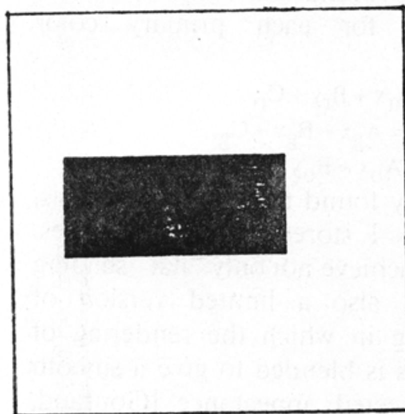
a) Zmin values and Enable flags before visibility calculations,

b) Enable flags and Z values for current polygon



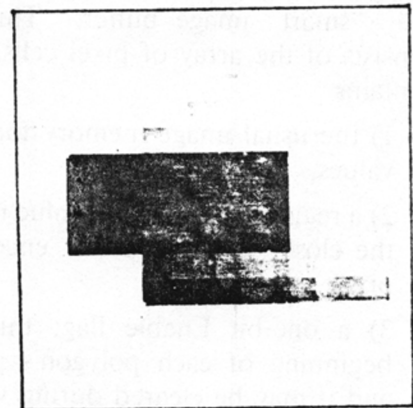
c) Enable flags and Zmin values after visibility calculations for current polygon.

Figure 4



a) Image buffer of one color before current polygon

b) Enable flags and f(x,y) of that color for the current polygon



c) Image buffer after current polygon's processing.

Figure 5

work on the problem (e.g., [Sutherland, et al., 1974], [Parke, 1980], [Weinberg, 1981], [Whitted, 1981]). We have structured our approach to fit easily into existing graphic systems and to be able to handle complex scene descriptions — ones with arbitrary many polygons. Such a goal argues for an approach in which polygons are processed sequentially and the image built up incrementally, rather than an approach in which the polygons are all buffered, sorted and the image generated all at once.

Further, this polygon-at-a-time approach dictates an image buffer ("frame buffer") in which the image is built up as the polygons are processed. Pixel-planes not only includes an image buffer, but enhances it with sufficient processing circuitry to do virtually the entire visibility and pixel painting process in the image buffer.

The steps needed to process each polygon are

a) identify all pixels that lie within the polygon,

b) determine the subset of visible pixels, that is, those not obstructed by some previous polygon, and

c) determine the proper color for each visible pixel.

The speed of Pixel-planes is achieved by having all these operations performed at every pixel simultaneously. The preprocessor receives the polygon, and broadcasts a series of commands to the "smart" image buffer. This image buffer consists of the array of pixel cells, each of which contains

1) the usual image memory for the RGB color values,

2) a register Zmin whose value is the distance of the closest polygon so far encountered at this pixel,

3) a one-bit Enable flag; this is set at the beginning of each polygon's pixel processing and it may be cleared during various stages of the pixel processing. A pixel stops responding to most commands except Reset once its Enable flag has been set to 0,

4) a one-bit comparator circuit, and

5) a one-bit full adder.

The central simplification is that all the above three steps can be performed by computing variations of the expression $f(x,y) = Ax + By + C$ in which x,y is the address of a pixel in the image.

The above three steps, a, b, c, are achieved by the following computations:

a) To determine the pixels that are inside the current polygon, the preprocessor broadcasts the coefficients A,B,C for each edge of the polygon. Any negative $f(x,y)$ sets its pixel's Enable flag to 0. Since each polygon is convex, a pixel is inside if and only if $f(x,y)$ is positive for all edges. Operationally, a pixel is inside a polygon if and only if its Enable flag is 1 after all the edges have been broadcast (Figure 3).

b) To determine the visible pixels, the preprocessor broadcasts the coefficients of the plane of the current polygon, i.e., $z = f(x,y) = Ax + By + C$. Each pixel whose Enable flag is still 1 compares its $f(x,y)$ with the value of its Zmin register. The pixel is visible if and only if $f(x,y) < Zmin$. Enabled pixels with $f(x,y) >= Zmin$ set their Enable flag to 0. The preprocessor rebroadcasts the Z coefficients, A, B, C, so that the still-enabled pixels can store the new Zmin values (Figure 4).

c) To determine the proper color for each pixel the preprocessor broadcasts three sets of coefficients, one for each primary color component, so

$Red = f_r(x,y) = A_r x + B_r y + C_r$,

$Green = f_g(x,y) = A_g x + B_g y + C_g$,

$Blue = f_b(x,y) = A_b x + B_b y + C_b$.

The pixels already found to be visible, that is, ones with Enable = 1, store the new RGB values. This process can achieve not only "flat" shading of polygons but also a limited version of "smooth" shading in which the rendering of adjacent polygons is blended to give a smooth rather than a faceted appearance [Gouraud, 1971] (Figure 5).

Pixel-planes Design

A naive implementation of the above concepts would have a multiplication and summation unit at each pixel. Fortunately there is a more compact way to implement the needed computations without significant loss of processing speed.

Figure 6a illustrates our basic computing structure, a node of a binary tree with a extra input coming from the side, similar to a serial multiplier [Lyon, 1976]. Both inputs and both outputs are bit-serial and synchronous with the system clock. The left descendent is the top input with a one clock cycle delay. The right descendent is the delayed sum of the top and side inputs. The node contains a carry flag so the two input values can be added in bit serial fashion.

With $C = c_k, c_{k-1}, ..., c_1, c_0$, input to the top and $A = a_k, a_{k-1}, ..., a_1, a_0$, input to the side, the left descendent produces C, the right descendent produces A+C. If we precede A with a single zero, in effect making it 2*A, the right descendent produces 2*A+C. If we combine three such nodes into a two-level tree as in Figure 6b, we obtain from the four output nodes, starting from the left-most node, C, A+C, 2A+C, 3A+C. If we build such a tree to n levels, input C on top and n-1 0's followed by A on the side, the 2^n outputs produce C, A+C, 2A+C, ..., (2^n-1)A+C.

Let us now consider the 2D grid of pixel cells forming the "smart" image memory. We place a binary tree of the above design above this grid with an output line above each column of pixel cells and run the line down through the columm. We note that if the pixel x,y addressing starts with 0 at the lower left pixel, the vertical line through each pixel contains the value Ax+C — although, of course, we have not input anywhere.

We place another copy of such a "multiplier" tree, rotated 90° counter-clockwise, to the left of the pixel array with a sequence of 0's input to its root and coefficient $B=b_k, b_{k-1},...,b_1,b_0$, preceded by n−1 0's, input to its "side". We run each output line horizontally through each row of pixel cells. The value on the horizontal lines, starting from the bottom one, will be 0, B, 2B, 3B, ..., (2^n-1)B. If we

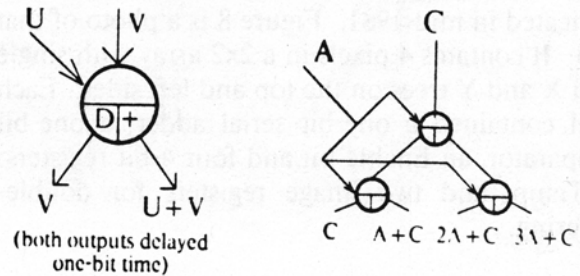


Figure 6
A multiplier tree
from one-bit serial adders

connect the horizontal and vertical lines running through a pixel cell to the inputs of the one-bit adder, its output will be (Ax+C)+By, the value we need for all our computation steps (Figure 7).

In summary then, the components of the design are

1) The preprocessor,
2) The X- and Y-multiplier trees, and
3) The smart image memory pixel cells.

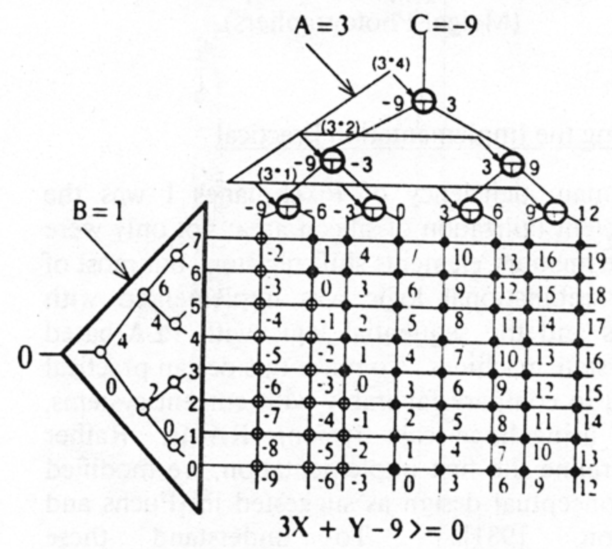


Figure 7
Calculating Ax+By+C
with two multiplier trees

Implementations

The first chip (Pixel-planes I) implementing these ideas was designed in December 1980 and fabricated in mid-1981. Figure 8 is a photo of that chip. It contains 4 pixels in a 2x2 array with single level X and Y trees on the top and left sides. Each pixel contained a one-bit serial adder, a one-bit comparator, an Enable bit and four 4-bit registers: Z, Temp, and two image registers for double-buffering.

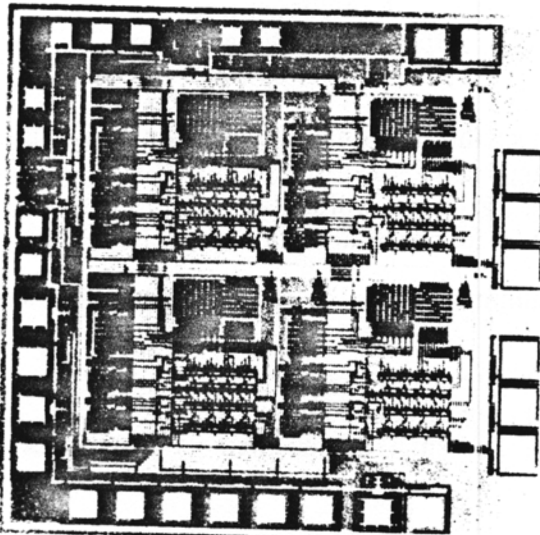


Figure 8
Photomicrograph of Pixel-planes I
(Melgar Photographers).

Making the Implementation Practical

The main deficiency of Pixel-planes I was the inefficient utilization of silicon area; not only were all the memory elements shift registers, but most of the combinational logic was implemented with PLA's and the sequential logic with PLA-based finite state machines. To make this design practical it had to compare favorably with current systems, those using large-scale dynamic RAMs. Rather than refine the first implementation, we modified the conceptual design as suggested in [Fuchs and Poulton, 1981]. To understand these modifications, let us consider, for example, the basic organization of a 4x4 pixel system (Figure 9).

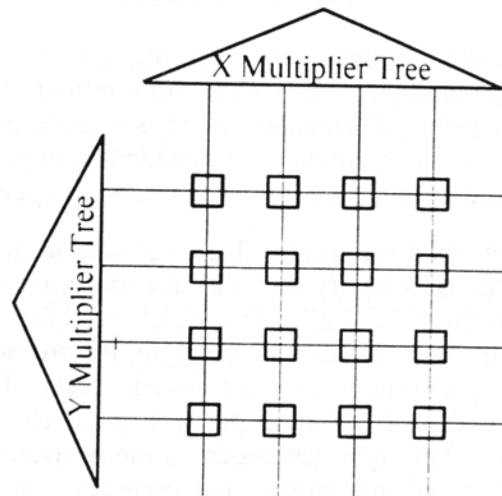


Figure 9
A 4 x 4 pixel system

We first switch C to come through the top of the Y-multiplier tree rather than the X-multiplier tree, a minor change. We next replicate the X-multiplier tree so there is one for each row of pixels, as in Figure 10.

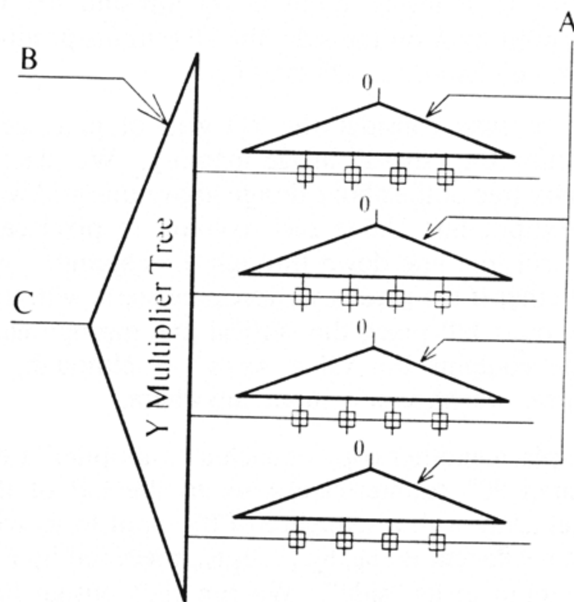


Figure 10
4 x 4 Pixel system
with duplicated X-multiplier trees

By reorganizing the multiplier trees, the adder can be removed from the pixel cells. Since all pixels within the same X-multiplier tree are adding the same value on their horizontal line, we take each horizontal line and connect it to the top of its own X-multiplier tree, as in Figure 11. The bits of A have to be input later than those of B and C to compensate for the travel time of By+C through the Y-multiplier tree. We next reorient the X-multiplier trees and their associated pixels so the entire system fits into a single column, as in Figure 12.

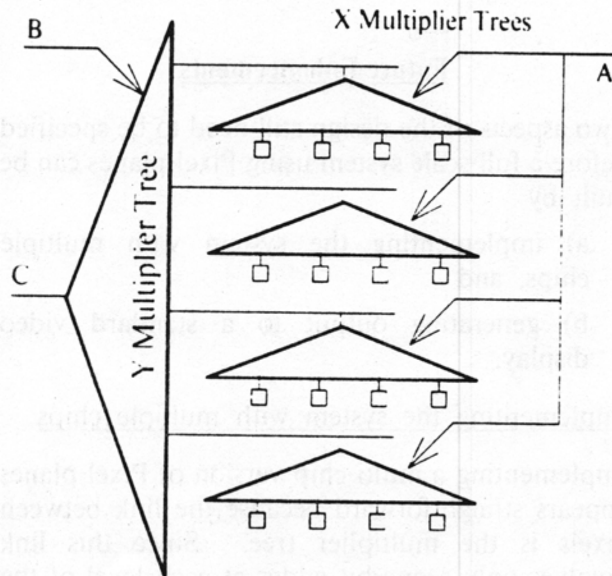


Figure 11
Rerouting Y-tree output to X-tree roots to eliminate the adder in each pixel cell

We now have a single one-dimensional tree that achieves a 2D 4x4 configuration by having the A coefficient (instead of B) going into the adders at the two lowest levels of the tree. In general, the configuration is determined by the selection of the coefficient for each level of the tree. (For instance, an 8 column by 2 row configuration is achieved by having A used at the lowest three levels of the tree.) Each adder now selects one of the two coefficients either by having one or the other "wired in" or according to the value of an "ABselect" flag.

The multiplier tree is implemented in a single column of nodes with the two outputs on the same side as the top input, as in Figure 13.

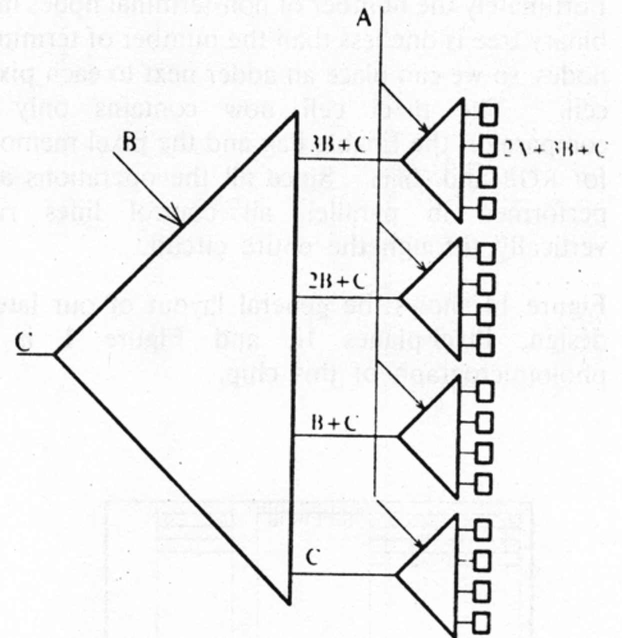


Figure 12
Layout of both X- and Y- trees along a single axis

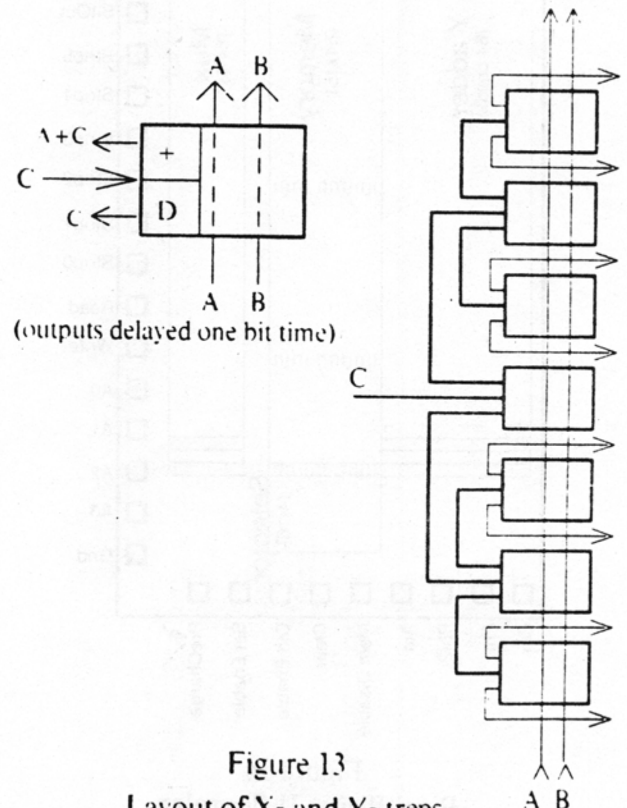


Figure 13
Layout of X- and Y- trees in a single column of nodes

Fortunately the number of non-terminal nodes in a binary tree is one less than the number of terminal nodes, so we can place an adder next to each pixel cell. The pixel cell now contains only a comparator, the Enable flag and the pixel memory for RGB and Zmin. Since all the operations are performed in parallel, all control lines run vertically through the entire circuit.

Figure 14 shows the general layout of our latest design, Pixel-planes II, and Figure 1 is a photomicrograph of this chip.

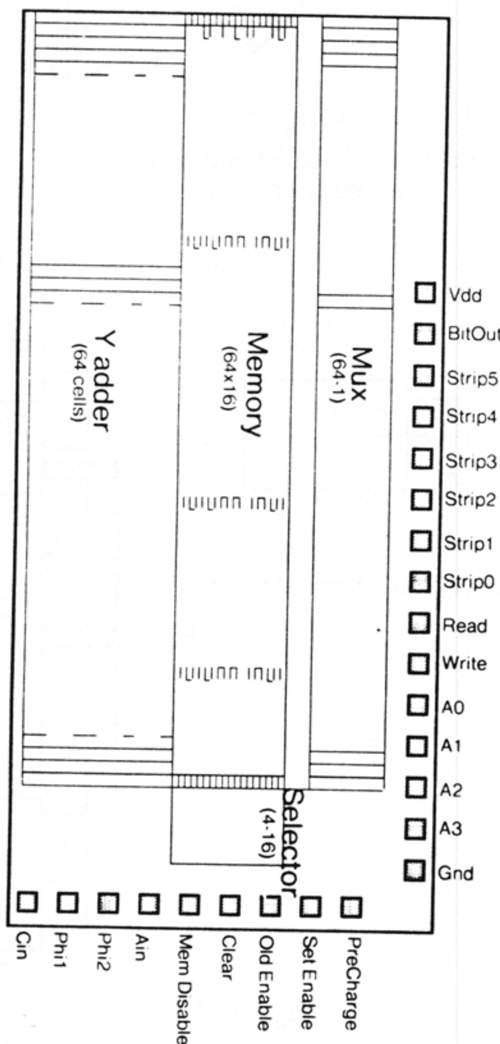


Figure 14
Pixel-Planes II floorplan.

Pixel-planes II Device Specifications

Pixel Cells: 64
Internal Memory: 1K dynamic RAM - 16 bits per Pixel Cell
Clocking: 2-phase with precharge
Estimated Static Power: 130mW @ 5.0 VDC (no pads)
Project Size: 1380λx2819λ, 2.7x5.6 mm ($\lambda = 2.0\mu$)
Project Area: 15.6 sq mm

Future Enhancements

Two aspects of the design still need to be specified before a full-scale system using Pixel-planes can be built by

a) implementing the system with multiple chips, and

b) generating output to a standard video display.

Implementing the system with multiple chips

Implementing a multi-chip version of Pixel-planes appears straightforward because the link between pixels is the multiplier tree. Since this link involves only a one-bit adder at each level of the tree, we replicate a "path" of the connecting tree in each chip, as in Figure 15

This path involves either a delay or a one-bit adder at each level, depending on the location of the chip in the interconnect tree. The conceptual realization of this on each chip is a row of the regular tree nodes, with a bit of a register selecting one or the other of the output branches for the next level's input (Figure 16).

The composite value in these one-bit registers specifies the x,y location of the chip's pixels in the overall image. This then is the address register of the chip. During system initialization, both this address register and the ABselect flags are loaded into each chip to determine the configuration of the individual pixels in each chip as well as the position of each chip in the overall image.

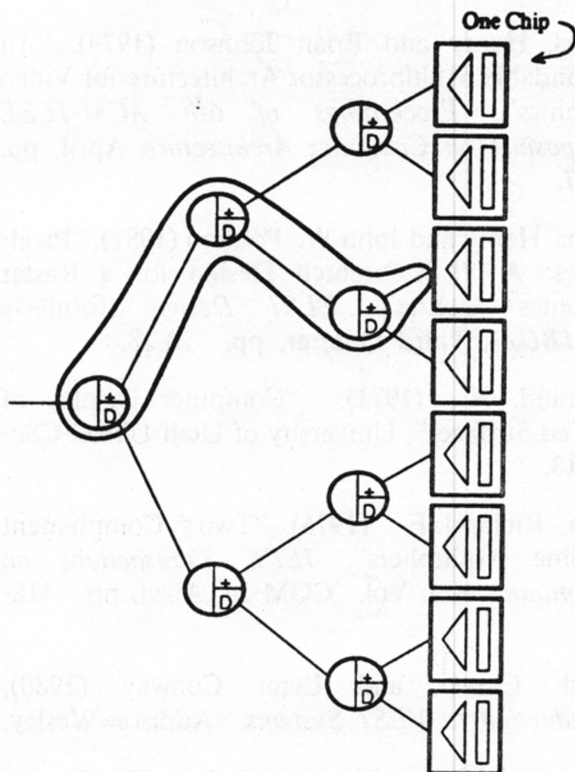


Figure 15
Interchip tree connecting an 8-chip system

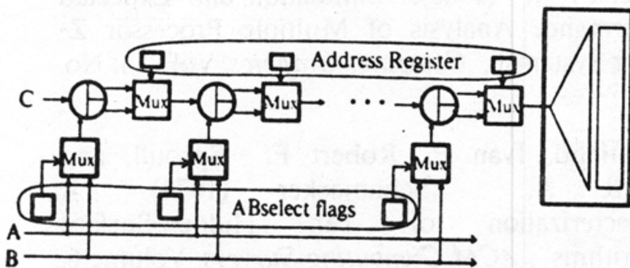


Figure 16
Chip address configuration control

Scan-Out Circuitry

The main timing constraint of all video graphic systems is the need to refresh the video screen at no less than 30 Hz. We foresee that chips in a large multi-chip system will each be configured as one tall column, an M by 1 pixel region. The advantage of this configuration is that during any one scan-line interval on the display, only one pixel is needed from each chip.

Meeting this requirement of reading out one pixel from each chip in approximately 63μS is not difficult, and we are currently considering a number of feasible alternatives.

Relation to Other Work

A number of systems have been constructed over the years for digital image generation. The commercial visual simulators of Evans and Sutherland, General Electric, and Singer-Link are each physically (and financially) immense systems with significantly more capabilities than the current Pixel-planes; many of them allow transparent polygons, perform anti-aliasing and simulate fog effects, for instance. We are working on including some of these effects in future versions of Pixel-planes.

In the realm of VLSI design for 3D graphics, James Clark's "Geometry Engine" [Clark, 1980], and Marc Hannah and Clark's Image Processors [Clark and Hannah, 1980] are designs that have been implemented. The Geometry Engine, being a transformation unit, would fit well into Pixel-planes, sitting between the graphic data base and the preprocessor. Their image processors, however, perform some of the same basic functions as Pixel-planes, but with fewer units. They generate lines and polygons in a sequential fashion similar to the distributed multi-microprocessor system described in ([Fuchs, 1977], [Fuchs and Johnson, 1979]). These designs are a significant improvement over single processor image generation systems, since each of the many processors can work largely independently. Since there may only be a few dozen processors in the system, however, each one is responsible for thousands of pixels and as such, larger polygons take longer to process. The system may not be able to keep up with the transformation module pipeline.

A recent design for an image processing (in contrast to image generating) machine [Blank et al., 1981] bears some similarities to the distributed processor nature of Pixel-planes. It is a SIMD (Single Instruction, Multiple Datastream) machine with a processor at every pixel and connections to its four neighbors. It differs from Pixel-planes in that it needs these connections for access to its

image data, thereby constraining both the physical layout of cells and the pinout as increasing number of cells are integrated into a single chip. Its processing task needs different capabilities for image processing, so the simple multiplier trees of Pixel-planes do not suffice for the processing circuitry.

Summary

We have described a raster graphic system for solving the long-standing computational bottleneck in the real-time digital image generation pipeline — the visibility and pixel painting calculations. The key concepts are

1) each pixel in the buffer can be enhanced with processing circuitry for doing most of its own processing,

2) the processing circuitry at each pixel can be simplified by formulating all the computations as $Ax+By+C$ for pixel address x,y,

3) calculating $Ax+By+C$ at all pixels can be done easily with a binary tree of one-bit adders,

4) the entire smart image buffer can be folded into a standard dynamic RAM array design by considering a pixel to be a column of bits in many words rather than a selected word within the RAM array,

5) multiple chip systems can easily be constructed out of identical chips by replicating on each chip a small amount of connecting circuitry.

References

Blank, Tom, Mark Stefik, and Willem vanCleemput (1981), "A Parallel Bit Map Processor Architecture for DA Algorithms", *ACM IEEE Eighteenth Design Automation Conference Proceedings*, Nashville, Tennessee, June 29–July 1.

Clark, James H. (1980), "Structuring a VLSI System Architecture", *LAMBDA*, 2nd Quarter.

Clark, James H. and Marc R. Hannah (1980), "Distributed Processing in a High-Performance Smart Image Memory", *LAMBDA*, 4th Quarter, pp. 40-45.

Fuchs, Henry (1977), "Distributing a Visible Surface Algorithm over Multiple Processors", *Proceedings of ACM Annual Conference.*

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

Fuchs, Henry and John W. Poulton (1981), "Pixel-planes: A VLSI-Oriented Design for a Raster Graphics Engine", *VLSI Design* (formerly *LAMBDA*), Third Quarter, pp. 20-28.

Gouraud, H. (1971), "Computer Display of Curved Surfaces", University of Utah UTEC-CSc-71-113.

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

Mead, Carver and Lynn Conway (1980), *Introduction to VLSI Systems.* Addison-Wesley.

Newman, William M. and Robert F. Sproull (1979), *Principles of Interactive Computer Graphics.* McGraw-Hill, 2nd edition.

Parke, F. I. (1980), "Simulation and Expected Performance Analysis of Multiple Processor Z-Buffer Systems", *Computer Graphics*, Vol. 14, No. 3.

Sutherland, Ivan E., Robert F. Sproull, and Robert A. Schumacker (1974), "A Characterization of Ten Hidden-Surface Algorithms", *ACM Computing Surveys*, Volume 6, Number 1.

Weinberg, Richard (1981), "Parallel Processing Image Synthesis and Anti-Aliasing", *Computer Graphics*, Vol. 15, No. 3.

Whitted, Turner (1981), "Hardware Enhanced 3-D Raster Display System", *Proceedings of the 7th Canadian Man-Computer Communications Conference*, Waterloo, June 10-12. ■