Fuchs85

Proceedings IEEE International Conference On Computer Design: VLSI in Computers, ICCD' 85, Rye Town Hilton, Port Chester, NY, October 7-10, 1985.

PARALLEL PROCESSING IN PIXEL-PLANES, a VLSI logic-enhanced memory for raster graphics

Henry Fuchs and John Poulton

Department of Computer Science
University of North Carolina
Chapel Hill, NC 27514

Abstract

We give a brief overview of Pixel-planes and discuss the ways in which the system uses parallelism to achieve its high performance. Some of these methods are widely used in various other VLSI logic-enhanced memory systems; at least one--the Linear Expression Evaluation tree--seems to be unique to Pixel-planes.

Pixel-planes Overview

This paper contains only a brief overview of the system. [Poulton et al. 85] contains a more complete description of the hardware, including details of the running prototypes; [Fuchs et al. 85] contains more details about the algorithms.

Figure 1 is a conceptual block diagram of a conventional graphics system that can support interaction with 3-D images. In such a system, a user interacts with a 3-D scene stored in a database, typically consisting of a list of vertices of polygons that tile the surfaces of objects in the scene. Real-time interaction depends on computation in a processing pipeline whose elements perform the following tasks:

1) Transform the scene according to the user's position and orientation; this step produces a list of polygon vertices in 'eye' coordinates.

2) Clip away parts of objects that are outside the field of view.

3) Scale for perspective by drawing closer together objects that are farther away; the result is a list of polygon vertices whose coordinates are given in 'screen' coordinates.

4) Color each polygon vertex according to a lighting model that takes into account the position, intensity, and color of user-specified light sources.

5) Scan-convert each polygon, determining which pixels are inside that polygon.

6) Remove hidden surfaces by determining which pixels in the current polygon are obscured by previously processed polygons.

7) Color each pixel, interpolating between vertex colors.

8) Store pixels in a frame buffer, from which the system can

9) Refresh the video display.

Steps (5)-(7) clearly represent the performance bottleneck in current graphics systems. Steps (1)-(4) require calculations whose number is only of the order of the number of polygons in the scene. Many current, affordable graphics systems can handle polygon transformations sufficiently rapidly to support real-time interaction with complex 'wire-frame' images or with a restricted class of 'flat-shaded' polygonal images. There exists, however, no affordable solution to the problem of rendering highly realistic, fully rendered 3-D scenes, perhaps with image enhancements such as smooth shading, anti-aliasing (to remove pixel artifacts), shadows, textures, transparent surfaces, fog effects, and so forth.

The Pixel-planes design attacks this problem by replacing the conventional rasterizer/frame buffer with a 'smart' frame buffer that not only *stores* a digital image but also performs much of the calculation needed to *generate* the image. This 'smart' frame buffer is built from custom chips in which conventional memory circuits are combined with

----

This research was supported in part by the Defense Advanced Research Projects Agency Contract No. DAAG29-83-K-0148 (monitored by US Army Research Office, Research Triangle Park, NC) and the National Science Foundation Grant No. ECS-83-00970.

some processing circuitry that allows computations to be carried out in parallel for all pixels in the display.

The processing circuitry in the enhanced memory chips can perform two kinds of operations:

1) Evaluate linear expressions of the form $F(x,y) = Ax + By + C$ simultaneously for all pixel locations (x,y). A,B,C are data broadcast to the memory chips.

2) Perform pixel-local arithmetic and logical operations on data stored at the pixel and on the linear expressions.

The strategy for applying this system to a given graphics image generation problem is to recast the problem into a form that requires only linear-expression evalution and pixel-local operations. To make use of these operations, the Translator (see Figure 1) converts the conventional description of graphics primitives (e.g., a list of vertices for each polygon) into the form of coefficients A,B,C and instructions for the pixel-local processors. The Image Generation Controller converts these coefficients and instructions into bit-serial form and broadcasts them to the array of enhanced memory chips.

A basic set of graphics operations for rendering convex polygons is described below. (Algorithms for shadows, spheres, textures, anti-aliased edges, etc. are described in [Fuchs et al. 85].)

Scan-conversion. At the beginning of each polygon, all pixels are enabled. For each edge of a polygon, the Translator calculates the coefficients for an expression $F(x,y)=Ax+By+C$, where $F(x,y)=0$ defines the points that lie along a line connecting two adjacent vertices, and broadcasts A,B,C and an 'edge' instruction to the array of enhanced memory chips. Each pixel-processor examines the sign of its value for the expression F. If negative, the pixel is outside the current polygon and is disabled for further processing. If positive, the pixel remains enabled. As each succeeding edge is processed, pixels are disabled by half-planes, until, after all edges are processed, only those inside the polygon remain enabled.

Hidden-surface Elimination. The Translator calculates a set of coefficients for the expression $z=F(x,y)=Ax+By+C$, the planar equation for the current polygon's surface, and broadcasts A,B,C and z-compare instructions to the memory chips. Each pixel maintains the z-coordinate of the closest visible polygon so far processed. Each pixel compares the value of the linear expression with its stored z; if the new z is farther away than the stored z, t
disabled (the new polygon is obscured at th
closer, the pixel remains enabled.

Smooth Shading. The Translator calc
each color, coefficients for the e
Intensity=Ax+By+C and broadcasts these c
and 'color' instructions to the memory
multi-sided polygon is at this stage b
triangular patches for shading. For each
for each primary color, enabled pixels u
color-intensity buffers with the new color i

As mentioned above, a number of other
have been developed, including castin
drawing and shading spheres, painting te
anti-aliasing edges [Fuchs et al. 85]. Figu
illustrate the results of some of these algori

Most of the system's processing power d
Pixel-planes's novel Linear Expression
(LEE), a binary-tree structure distributed
over an array of identical enhanced memo
form the 'smart' frame buffer. The prin
LEE is illustrated in Figure 4. Each node
takes in a serial bit-stream at its 'top' inpu
unchanged, but delayed by one clock c
left-hand output. The right-hand output i
adding a second bit-stream from the '
bit-serial addition injects a one-cycle dela
branch and requires the usual local ca
shown in the figure. In effect, each tree
both possible values of a partial product
to Ax + C. Because of the leading 0's i
stream, A_0 arrives at the bottom level of th
same time as C_0; thus, 1*A is added at eac
bottom level. At the second level, A_0 is
with C_1, so that 2*A is added at this level
In general an n-level tree can generate
values for an expression of the form Ax+C

To generate the full linear expression f
with 1024x1024 pixels, a 10-level X-tre
to generate the 1024 distinct values of
outputs of this tree in turn feeds the 'top' i
Y-trees, and this ensemble generates the
values of Ax+By+C. Note that the cost i
for this entire structure is only a singl
(three one-bit registers and one bit-seri
each pixel, but the performance is equiva
20-stage serial multiplier at every pixel.

A simplified block diagram of the
memory chip is shown in Figure 5. In ad

Linear Expression Evaluator, these chips contain an array of tiny bit-serial ALU's that perform pixel-local operations (one ALU for each pixel), the pixel memory itself, and circuitry for scanning out data to refresh a display. Only a small subset of the pixels in a display can be placed on a single chip, so the LEE is easily distributed over many chips. This is done by building a subtree that covers only the pixels on the chip (organized as a vertical column); the remainder of the tree is implemented by circuitry that maps a path through the complete tree. Called a 'supertree', this path is programed during system intialization, loading each chip with its pixel-column's address in the display. The supertree construct considerably simplifies the system-level implementation of our system at little cost (about 5%) in silicon area. In our current chips, about 18% of the circuit area is devoted to the LEE, about 12% to the ALU array, and the remaining 70% is memory.

Uses of Parallelism

C. Seitz in his survey of concurent VLSI architectures [Seitz 84] notes that logic-enhanced memory systems "achieve rather remarkable performance per cost, in comparison to the same computation on a general purpose sequential computer, by 1) specialization of the system to the algorithms, 2) concurrent operation of an appreciable subset of the nodes in the system, 3) localization of communication between the stored data and the logic in the node (an unlimited storage bandwidth), and 4) localization of communication in the connnection plan between nodes." All these methods are used by Pixel-planes:

1) Pixel-planes is highly specialized to its application, graphic operations on frame buffer memories. It is tuned to a rather general class of algorithms most of whose operations can be expressed as linear functions of the pixel's address on the screen. We note, however, that Pixel-planes is not restricted to any particular set of algorithms, as several outside researchers have already devised clever algorithms for the system.

2) Concurrent operation of a large subset of the nodes in the system normally takes place, since all the pixel processors can be operating at the same time, in SIMD fashion. Certain algorithms, of course, exploit this parallelism better than others. Shadow volume calculations, for example, will keep a significant fraction of the pixel processors busy (doing useful work) if the shadow of a polygon is cast throughout much of the image. On the other hand, processing small polygons is often inefficient since for much of the processing time, most of the pixel processors are idle . If the chips were to be decoupled with some FIFO buffers, however, different polygons can be processed in different chips [Poulton et al 85]. Simulations indicate that speedups of 5X or more could be achieved for even simple scenes with about 1000 polygons.

3) Pixel-planes achieves "localization of communication between stored data and logic in the node" by having each pixel processor's memory physically adjacent to the processor. A pixel's memory can only communicate with its processor and the video scanout control. Similarly a processor can only receive data from the Linear Expression Evaluator tree or its memory. It can only write data into its own memory. In addition, since the entire machine operates on the same instruction stream, all the pixel memories address the same bit at any given cycle, so space efficiency is gained by having a single common memory control for the entire chip.

4) Communication between nodes is a massive problem in most concurrent systems. W.D. Hillis [Hillis 81] notes that "the most difficult technical problem in constructing a connection memory is the communications network." Pixel-planes is fortunate to be dealing with a class of applications that do not rely on node-to-node communications, so **no** such communications are included in the machine. Although we have considered several communication mechanisms, we have not found any to be necessary.

We conclude this discussion by restating our earlier point that most of the system's processing power comes from its novel Linear Expression Evaluator. If the modules in the system in front of the memory chips (the translator and controller) could operate rapidly enough to take advantage of it, we'd consider adding another LEE tree on the chips. Many of the algorithms could be speeded up by having multiple linear expressions evaluated concurrently -- multiple edges of a polygon, for instance. Similarly, multiple color primaries (red, green, blue) could be calculated at the same time, multiple shadow volume planes, etc.

Conclusions

Pixel-planes achieves its execution efficiency by a variety of techniques, most of which are also applicable to other logic-enhanced memory systems. Although its algorithms could be executed on many fine-grained multiprocessors with enough nodes to allocate one per pixel, Pixel-planes achieves its dramatically faster performance by using an additional structure, a binary tree (the LEE) for parallel, pipelined evaluation of its often-needed linear expressions.

References

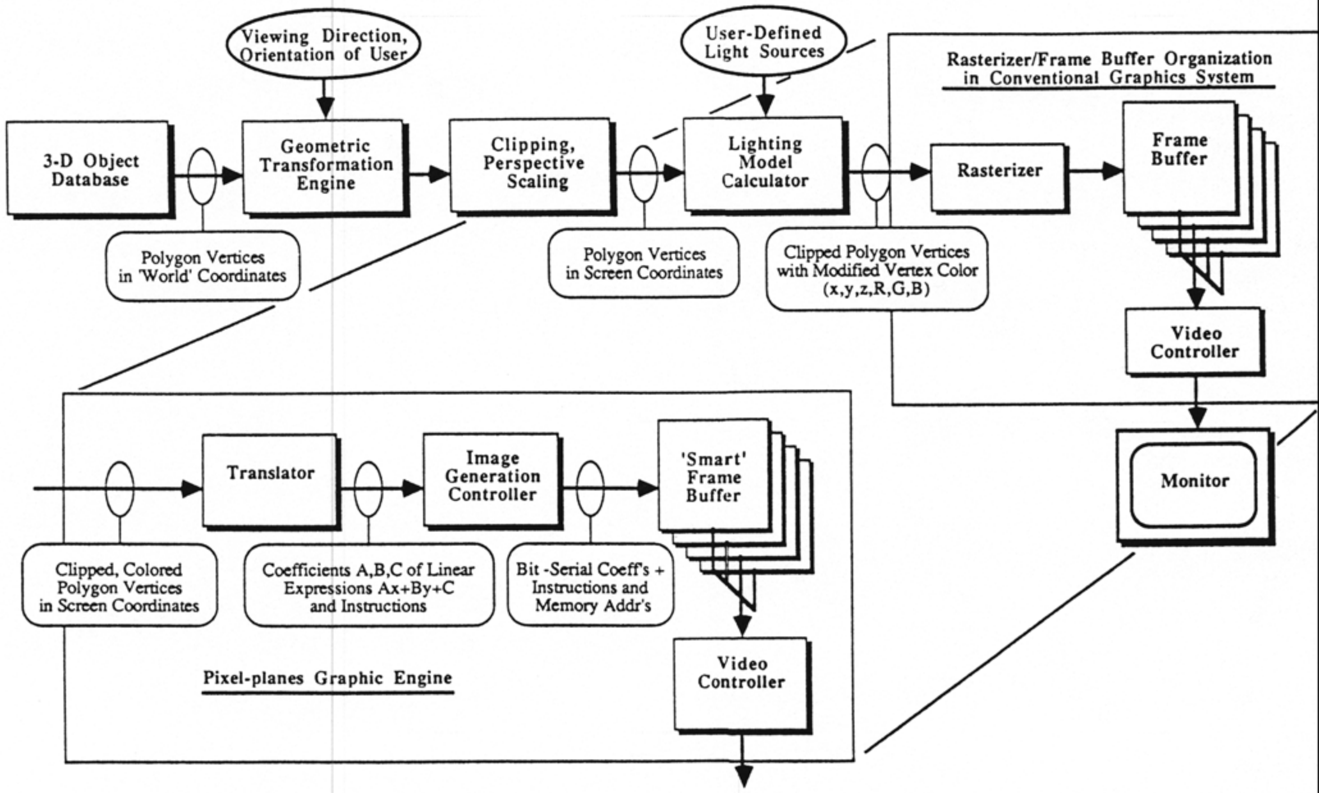
Fuchs, H., J. Goldfeather, J.P. Hultquist, S. Spach, J.D. Austin, F.P. Brooks, J.G. Eyles, J. Poulton, "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements with Pixel-planes", Computer Graphics, Vol. 19., No. 3, July 1985 (Proceedings, Siggraph '85).

Hillis, W.D., "The connection machine (computer architecture for the new wave," Massachusetts Inst. Technol., Cambridge, AI Memo 646, Sept 1981.

Poulton, J., H. Fuchs, J.D. Austin, J.G. Eyles, J Heinecke, C-H Hsieh, J. Goldfeather, J.P Hultquist, and S. Spach, "Pixel-planes: Building VLSI-Based Graphic System," Proceedings o the 1985 Chapel Hill Conference on Very Larg Scale Integration, Computer Science Press Rockville, MD.

Seitz, C.L., " Concurrent VLSI Architectures," IEEE Transactions on Computers, Vol. C-33, No. 12 December 1984.

* *



Viewing Direction, Orientation of User
User-Defined Light Sources
Rasterizer/Frame Buffer Organization in Conventional Graphics System
3-D Object Database
Geometric Transformation Engine
Clipping, Perspective Scaling
Lighting Model Calculator
Rasterizer
Frame Buffer
Polygon Vertices in 'World' Coordinates
Polygon Vertices in Screen Coordinates
Clipped Polygon Vertices with Modified Vertex Color (x,y,z,R,G,B)
Video Controller
Monitor
Translator
Image Generation Controller
'Smart' Frame Buffer
Clipped, Colored Polygon Vertices in Screen Coordinates
Coefficients A,B,C of Linear Expressions Ax+By+C and Instructions
Bit -Serial Coeff's + Instructions and Memory Addr's
Video Controller
Pixel-planes Graphic Engine

Figure 1: Relationship between Pixel-planes Graphics Engine and a conventional system for rendering 3-D images. The Engine replaces rasterizer and frame buffer with a 'smart' frame buffer built from custom, logic-enhanced memory chips.

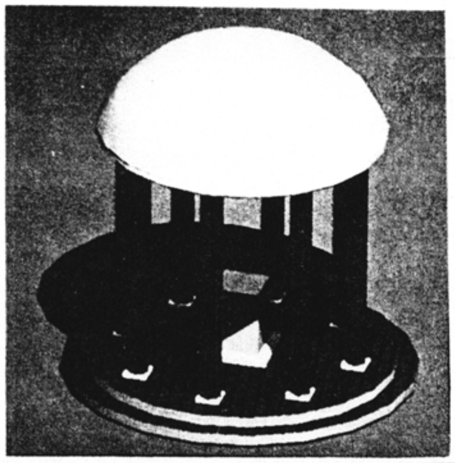


Figure 2: The Chapel Hill "Old Well", smooth shaded and with true shadows (simulation). Input is list of 357 polygons; estimated time with 10MHz system is 13.8 msec without textures, 14.3msec with textures (66 textured polygons).

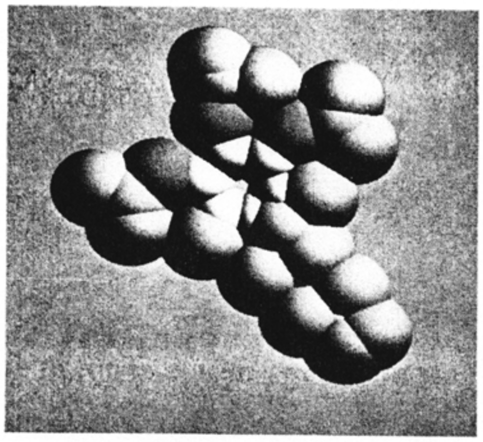


Figure 3: Simulated image by Pixel-planes functional simulator of Trimethoprim molecule containing 40 side-lit spheres. Estimated time for 10MHz system is 1.7msec. (Data courtesy of Dr. Michael Corey and Lee Kuyper of Burroughs Wellcome and Michael Pique and Doug Schiff of UNC.)

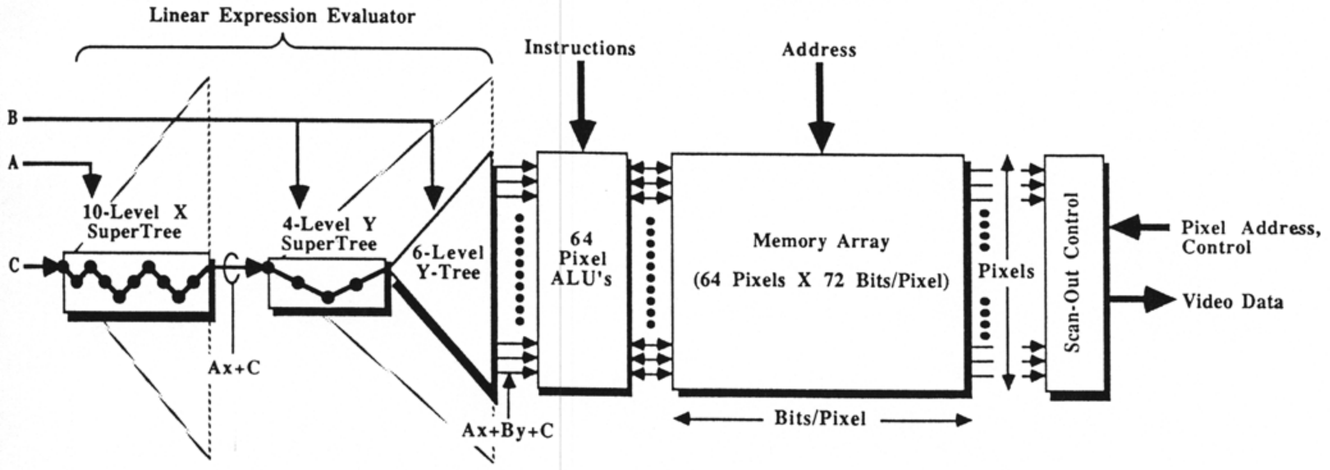


Figure 4: Fundamental operation of the Linear Expression Evaluator.

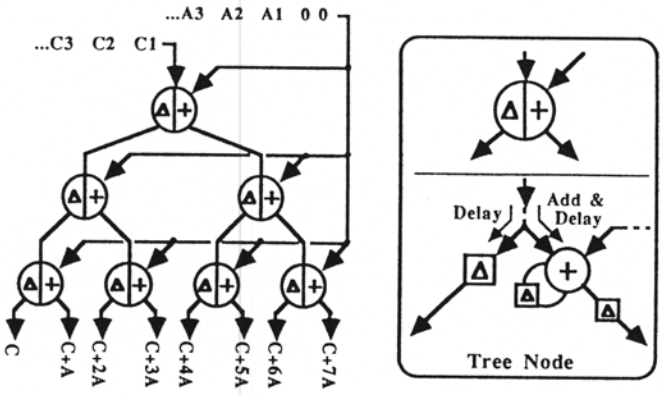


Figure 5: Block diagram of Pixel-planes enhanced memory chips.