PIXEL-PLANES: BUILDING A VLSI-BASED RASTER GRAPHICS SYSTEM

Technical Report 85-001; Revised March 14, 1985

John Poulton, Henry Fuchs,
John D. Austin, John G. Eyles, Justin Heinecke,
Cheng-Hong Hsieh, Jack Goldfeather,*
Jeff P. Hultquist, Susan Spach

The University of North Carolina at Chapel Hill
Department of Computer Science
New West Hall 035 A
Chapel Hill, N.C. 27514

**Department of Mathematics, Carleton College, Northfield, MN, on sabbatical at Department of Mathematics at University of North Carolina at Chapel Hill*

PIXEL-PLANES:
Building a VLSI-Based Graphics System*

John Poulton, Henry Fuchs, John D. Austin, John G. Eyles
Justin Heinecke, Cheng-Hong Hsieh
*Jack Goldfeather***, *Jeff P. Hultquist, Susan Spach*

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

1. Introduction

Pixel-planes is a VLSI-based raster graphics machine that will support real-time interaction with three-dimensional shadowed, shaded, and colored images. The system's cost and complexity will be comparable to present-day line drawing systems, making it suitable for use with high-performance workstations. Potential applications include computer-aided design, medical display and imaging, molecular modeling, and simulators for flight and navigational training.

The fundamental ideas in this design have been previously published [Fuchs and Poulton, 1981; Fuchs *et al.*, 1982]. This paper reports recent progress toward building a full-scale working *Pixel-planes* system, development of a number of new graphics algorithms for the machine, and refinements in system architecture and design methods.

Much of current research in experimental graphics systems is aimed at improving the speed of image generation by dividing the display into small regions, each of which is handled by separate concurrent processors [Clark and Hannah, 1980; Gupta *et al.*, 1981; Demetrescu, 1985]. In *Pixel-planes*, this division is imbedded in a binary tree that performs the bulk of the system's computations and distributes the results to all pixels. Each pixel consists of an array of memory elements and a small processor that only performs operations local to the pixel. The heart of the system is a Smart Frame Buffer consisting of an array of identical custom chips that contain the binary tree, pixel memories and processors, and video scan-refresh circuitry. These enhanced memory chips employ a moderately dense, conventional dynamic RAM that takes up about 2/3 of the chip's silicon area; the processing circuitry takes up the remaining 1/3.

The fundamental operation of the *Pixel-planes* system is calculating linear expressions of the form $Ax + By + C$ where x and y are the coordinates of a pixel and A, B, and C are data inputs to the system. These expressions are calculated bit-serially in a binary tree multiplier/accumulator, simultaneously for all pixels. The system's hardware is not built to execute a specific set of graphics algorithms. Instead, many different algorithms can be recast into forms that evaluate linear expressions and/or require only pixel-local operations. We are continually surprised at the variety of algorithms that we and others are able to express in this form, and it is clear that the architecture is much more powerful and more general than we had first imagined.

---

* To appear in the Proceedings of the 1985 Chapel Hill Conference on VLSI, May 15–17, 1985. This research supported in part by the Defense Advance Research Project Agency Contract number DAAG29-83-K-0148 (monitored by U.S. Army Research Office, Research Triangle Park, NC) and the National Science Foundation Grant number ECS-8300970.

** Department of Mathematics, Carleton College, Northfield, MN, on sabbatical at Department of Mathematics at University of North Carolina at Chapel Hill.

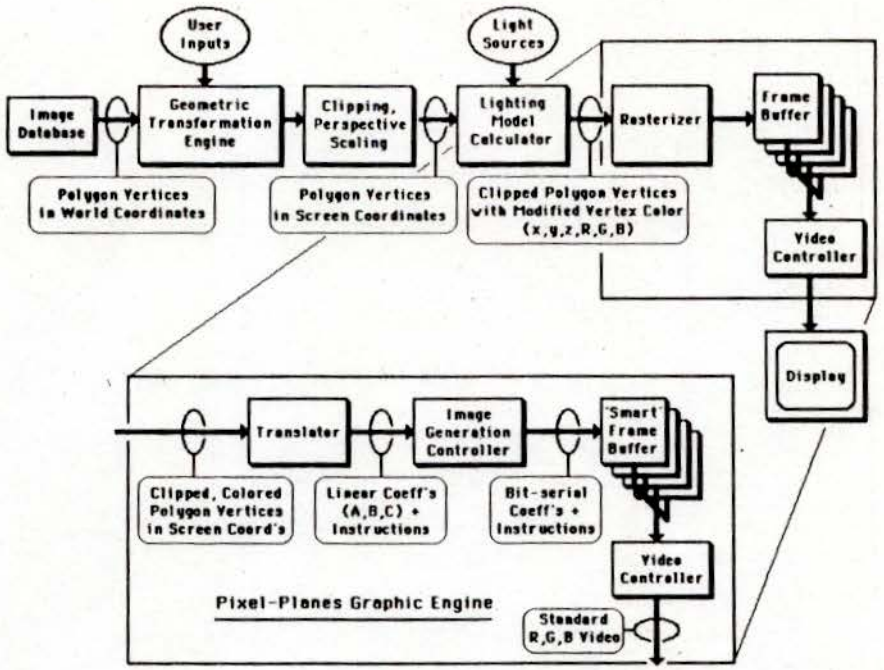

Figure 1: Pixel-Planes Graphics Engine replaces the rasterizer, frame buffer, and video controller in a conventional graphics system.

2. Pixel-Planes Graphics System

2.1 System Overview

Figure 1 shows the relationship between the *Pixel-planes* graphics system hardware and a conventional color graphics system.

The 'front end' of the conventional graphics system is a pipeline of special processors that manipulates an image database. The database contains (typically) a list of polygons that tile the surfaces of the objects in a scene. Each polygon is described as a list of vertex coordinates (x, y, z in 'world' coordinates) and colors (values of *Red*, *Green*, *Blue* that specify the intrinsic color of the vertex). A transformation engine operates on the coordinates of the vertex list for each polygon, transforming the polygon to 'eye' coordinates in response to user input from joystick, trackball, or some similar device. Next, polygons (or portions of polygons) that are outside the viewing pyramid are clipped and perspective division is performed to transform 'eye' coordinates to 'screen' coordinates. Finally, a lighting model calculator modifies each vertex's intrinsic color according to the position and intensity of light sources. The output of the front-end pipeline is still a list of polygon vertices, but with vertex coordinates and colors transformed to the proper value for display.

In advanced color graphics systems, the rasterizer performs a series of steps needed to translate a list of polygon vertices into a smooth-shaded, rendered, digital image, with hidden surfaces properly removed, and perhaps anti-aliased to reduce pixelization artifacts. In general, these calculations must be performed for every pixel for every polygon processed, implying massive amounts of computation and very large memory bandwidth.

The *Pixel-planes* Graphics Engine replaces the rasterizer, frame buffer, and video controller of a conventional system. Its main component is a Smart Frame Buffer composed of custom VLSI enhanced memory chips. It addresses the computational problem with a highly parallel processor that mimics a processor per pixel. The memory bandwidth bottleneck is overcome by intimately

connecting processing circuitry and memory.

2.2 Pixel-Planes Graphics Engine

The components of the Engine are:

The Translator, a special purpose micro-programmable floating-point computer, converts the scene description from a polygon vertex list into the form of coefficients A, B, C of the linear expression $F(x,y) = Ax + By + C$. It also produces an encoded instruction for each step in processing polygons or other primitives (*e.g.*, edge, z-compare, circle, paint-Red). Translation will involve, for example, describing an edge of a polygon in the form $F(x,y) = 0$, or specifying the polygon's planar surface in the form $z = F(x,y)$. In the system now under construction, the Translator is a 5 MFlop micro-programmable engine based on the Weitek 1032/1033 floating-point chip set.

The Image Generation Controller (IGC) converts the word-parallel, floating-point A, B, C coefficients from the Translator to bit-serial, 2's complement data, decodes each instruction into a stream of control words, and outputs this data and control along with the clock for the Smart Frame Buffer. Currently, the IGC is implemented as a custom chip that serializes the coefficient data and a micro-programmable control sequencer built using standard TTL parts.

The Smart Frame Buffer is organized as a series of 'logical boards', each with an array of enhanced memory chips, as shown in Figure 2. This organization reduces the bandwidth (pin-count, operating speed) necessary at the memory chip's video-data output port. Each logical board contains a 32-bit-wide register for video data, and successive logical boards are daisy-chained together to form a high-speed shift-train. Every L cycles (where L is the number of logical boards), shifting is disabled and the shift-train is loaded from a parallel set of registers on each board. While shifting is enabled, these parallel registers are loaded, one byte at a time, from selected memory chips.

Data, control, and clocks both for image generation and video output are broadcast to the enhanced memory chips. No data or control need be returned from the memories to the IGC or Video Controller, so the busses can easily be pipelined for high-speed operation.

In addition to these two uni-directional busses, a single serial scan-path links all memory chips in the frame buffer. During system operation, the scan-path takes the place of chip-address decoding, carrying a series of scan tokens that determine which set of memory chips is enabled for video output (only one chip on each logical board is enabled at one time). During system initialization, the scan-path is used to load various configuration registers, as discussed in Section 2.3.

The Video Controller is similar to those in conventional systems, with the exception of the token-passing method of addressing. The current version is capable of supporting a variety of display types (30 and 60 Hz, interlaced and non-interlaced, NTSC and non-standard) and any number of enhanced memory chips.

2.3 Enhanced Memory Chips

Figure 3 is a block diagram of the enhanced memory chip. It contains the Multiplier that implements the binary-tree linear-expression evaluator, an array of pixel ALU's, and a Memory system that stores data for each pixel and provides a video scan-out mechanism.

A conceptual model of a binary-tree multiplier/accumulator is shown in Figure 4. This structure is recognizable as a variation on the simple serial-parallel multiplier [Lyon, 1976], where both possible values of partial product are generated at each stage. If such a tree has N levels, and A contains K significant bits, A must be preceded by $(N-1)$ 0's; 2^N distinct values of $Ax + C$ will be generated ($0 <= x < 2^N$), each value being $N + K + 1$ bits long.

To generate the linear expression $Ax + By + C$, two binary-tree multiplier/accumulators are stacked one atop the other. For a system with 1024 x 1024 pixels, a 20-level tree is required. The

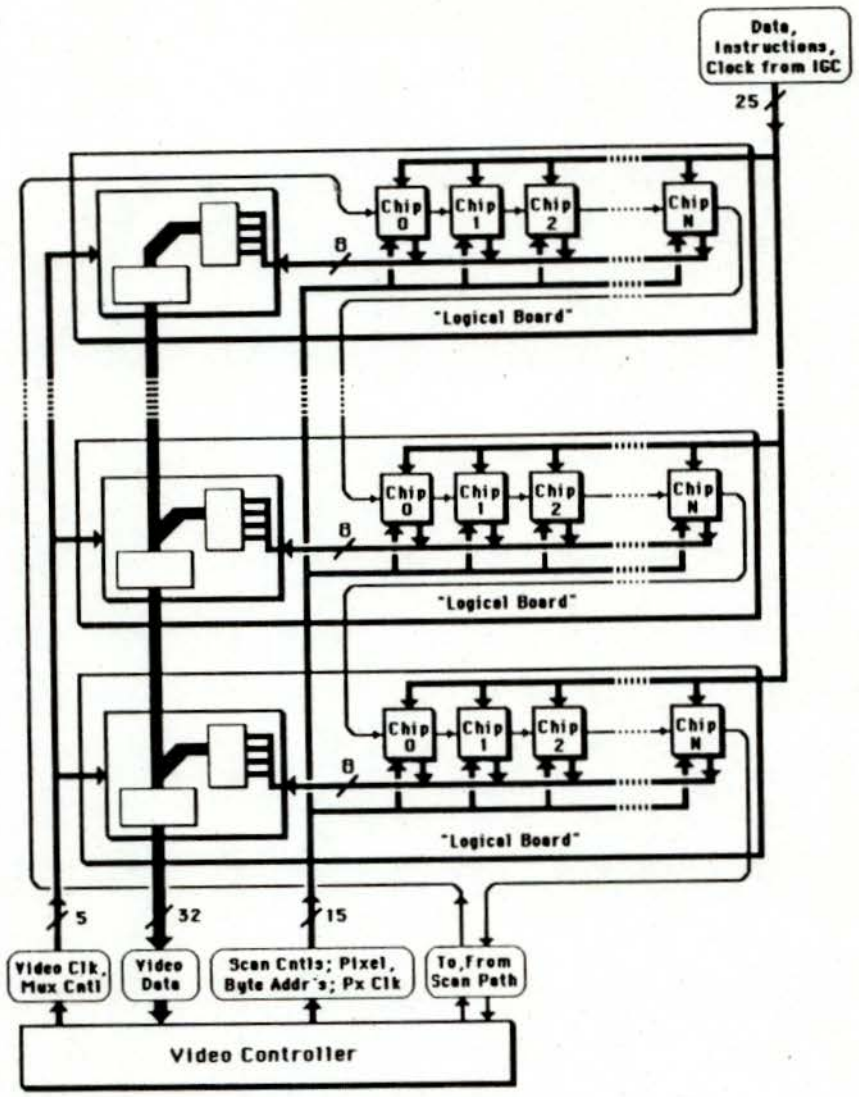

Figure 2: Pixel-Planes Smart Frame Buffer organization.

top 10 levels of the tree calculate the 1024 values of $Ax + C$. The bottom 10 levels can be thought of as 1024 subtrees, each of which receives one value of $Ax + C$ as its root input, gets B as its side input, and generates 1024 values of $Ax + By + C$. For a system with N x N pixels, the binary tree requires $N^2 - 1$ multiply/accumulate stages. It performs the same function, at the same speed, as a full $2N$-stage multiplier at every pixel (requiring $2N^2 logN$ stages).

Only a small fraction of the pixels in a display can be put on a single chip, so it is necessary to break the binary tree into multiple chips. This is done by implementing a small sub-tree on each chip that covers only the pixels on the chip. A 'supertree' on each chip implements the tree levels above the sub-tree. It contains one multiply/accumulate stage for each level above the sub-tree. As shown in Figure 5, registers in the supertree are loaded at system initialization to map a path through the full tree to the local subtree. This defines the position of the chip's 64-pixel column in the full image.

It is possible, of course, to design a system without supertrees. If each chip were equipped with one extra tree node whose outputs go off-chip, the tree levels above each local subtree could be completed using inter-chip wiring. This external wiring would, however, reduce system speed

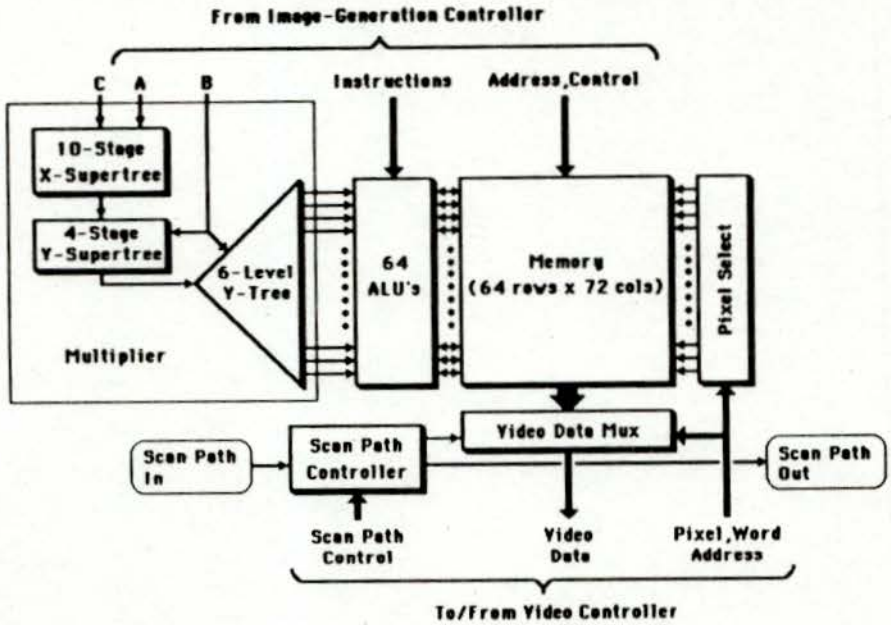

Figure 3: Block diagram of the Pixel-Planes memory chip.

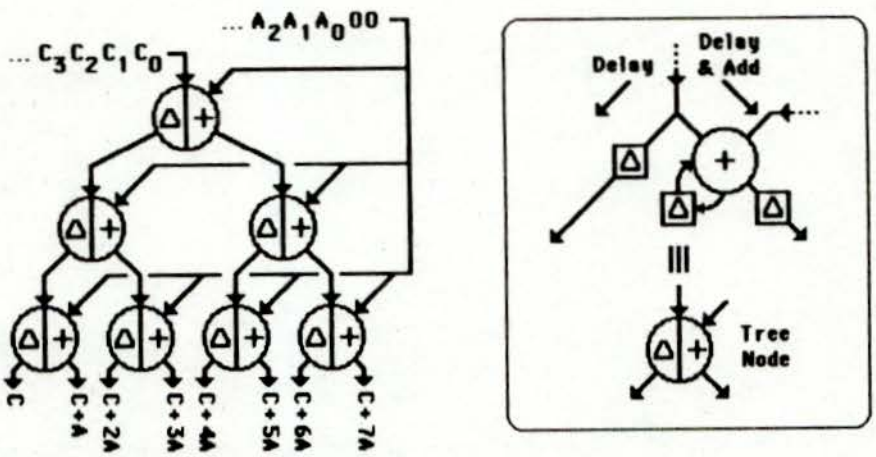

Figure 4: Conceptual model of binary-tree linear expression evaluator.

and complicate board-level construction. The configurable supertree on our current chips has 14 levels, requiring only another 14 multiply/accumulate stages and 14 registers—a relatively modest penalty in silicon area. It also makes possible a module-redundancy scheme, described below, that supports fault tolerance in our system.

Figure 6 shows the block diagram of the ALU at each pixel. Logical operations in the ALU are performed by a one-bit adder with a multiplexer/complementer on each of its three inputs. All ALU's in the system receive function-select and register-load controls broadcast from the IGC, so the ensemble of memory chips has SIMD concurrency. The ALU contains an Enable register that controls memory write access, allowing each pixel to determine locally whether current memory contents can be overwritten.

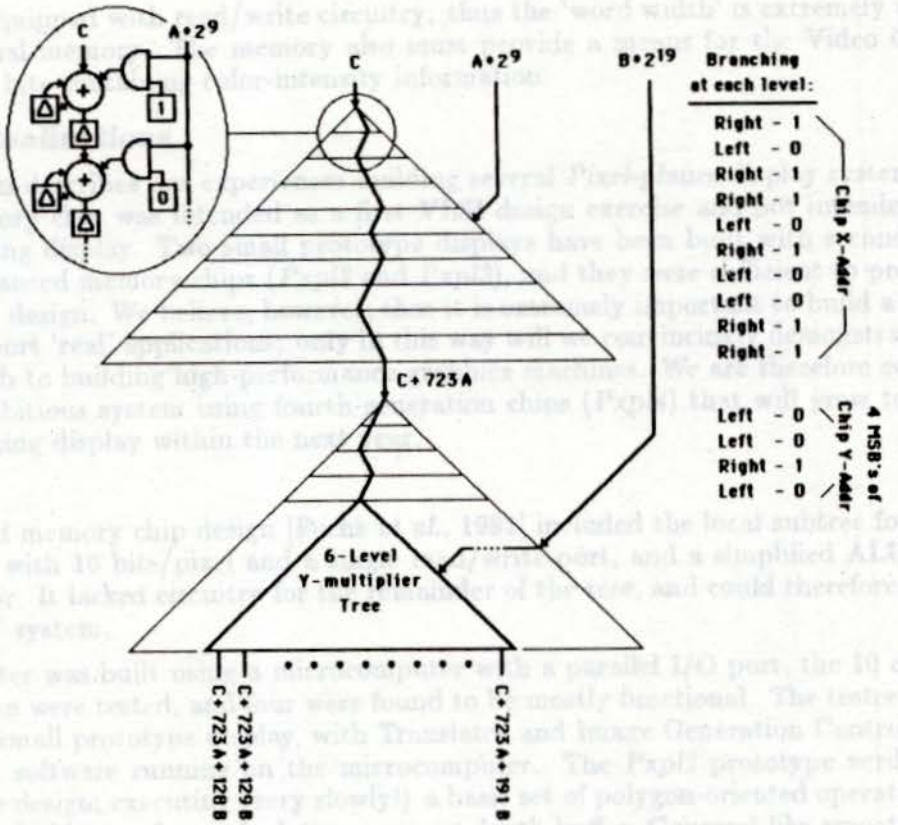

Figure 5: Supertree maps a path through full tree on each chip.

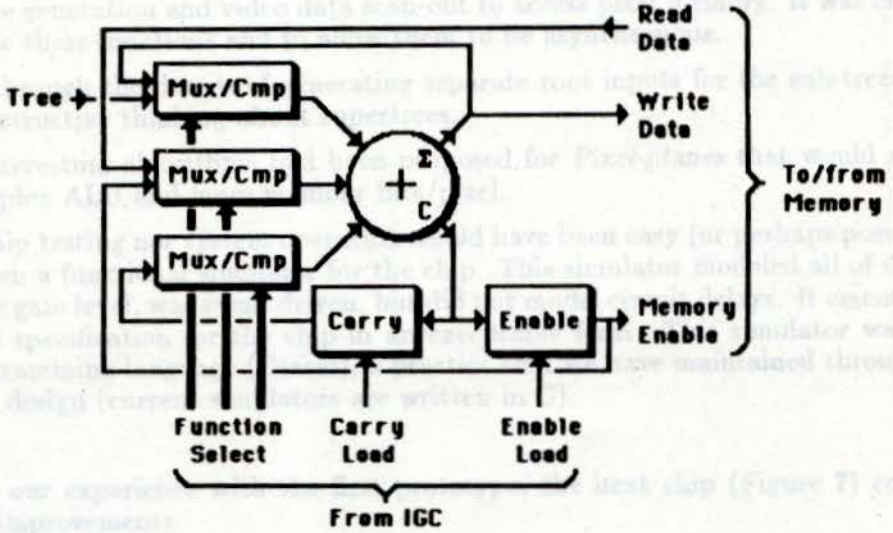

Figure 6: Block diagram of the pixel ALU.

The Memory system consists of a relatively dense dynamic RAM array. Each column of cells in the array contains corresponding bits in each pixel on the chip. Each row contains all of the bits in

a pixel and is equipped with read/write circuitry; thus the 'word width' is extremely wide relative to a conventional memory. The memory also must provide a means for the Video Controller to access memory bits containing color-intensity information.

3. System Realizations

This section describes our experiences building several *Pixel-planes* display systems. Our first enhanced memory chip was intended as a first VLSI design exercise and not intended to become part of a working display. Two small prototype displays have been built with second- and third-generation enhanced memory chips (*Pxpl2* and *Pxpl3*), and they were sufficient to prove the basic concepts in the design. We believe, however, that it is extremely important to build a system large enough to support 'real' applications; only in this way will we convincingly demonstrate the utility of this approach to building high-performance graphics machines. We are therefore constructing a much more ambitious system using fourth-generation chips (*Pxpl4*) that will grow to a full-scale, full-speed working display within the next year.

3.1 Pxpl2

Our second memory chip design [Fuchs *et al.*, 1982] included the local subtree for 64 pixels, a memory array with 16 bits/pixel and a single read/write port, and a simplified ALU with only a Carry generator. It lacked circuitry for the remainder of the tree, and could therefore only be used to build a 'toy' system.

A chip tester was built using a microcomputer with a parallel I/O port, the 10 chips received from fabrication were tested, and four were found to be mostly functional. The tester then became the host for a small prototype display, with Translator and Image Generation Controller functions carried out by software running on the microcomputer. The *Pxpl2* prototype verified the basic concepts in the design, executing (very slowly!) a basic set of polygon-oriented operations (polygon area definition, hidden-surface calculations using a depth-buffer, Gouraud-like smooth shading).

This exercise immediately suggested a number of design improvements:

(1) Since the memory had only a single port, image-generation had to be halted to refresh the display. This required a complex control mechanism with an external scan-line buffer to allow both image generation and video data scan-out to access pixel memory. It was clearly essential to separate these functions and to allow them to be asynchronous.

(2) Working through the details of generating separate root inputs for the sub-trees on each chip led to constructive thinking about supertrees.

(3) Several interesting algorithms had been proposed for *Pixel-planes* that would require both a more complex ALU and more memory bits/pixel.

Neither chip testing nor system operation would have been easy (or perhaps possible) if we had not first written a functional simulator for the chip. This simulator modeled all of the circuitry in the chip at the gate level, was event driven, but did not model circuit delays. It essentially captured the functional specification for the chip in an executable form. This simulator was written in a standard programming language (Pascal), a practice that we have maintained through the current version of the design (current simulators are written in C).

3.2 Pxpl3

Based on our experience with the first prototype, the next chip (Figure 7) contained many architectural improvements:

(1) A complete tree was included on each chip, implemented with the supertree notion described above.

(2) The ALU was modified to the form shown in Figure 6 to support a variety of new algorithms.

(3) Memory size was increased to 32 bits/pixel. We used a dual-ported memory cell to allow separate, asynchronous access to the pixel memory for scan refresh.

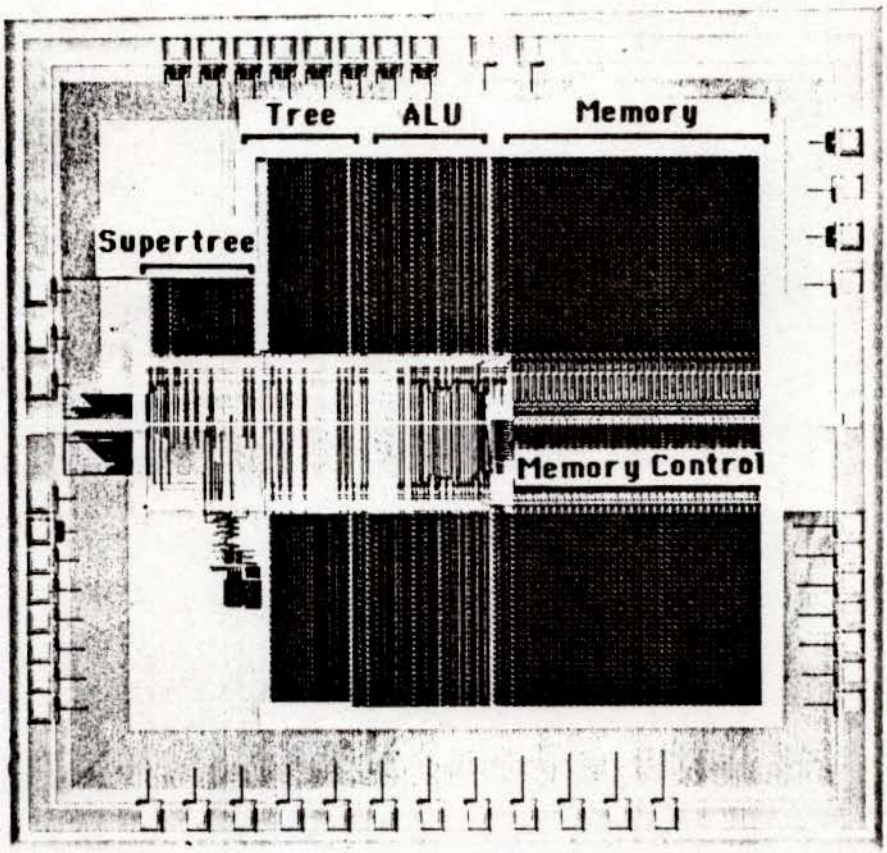

Figure 7: Photo of Pxpl3 memory chip showing major function blocks.

(4) Since memory access on the video-data port always proceeds in scan-line order, we installed a pixel-addressing mechanism that uses serial-shift tokens. A 'global' token that passes from chip to chip performs chip select, while a 'local' token register inside each chip manipulates the pointer to the currently selected pixel. The token-addressing scheme reduces chip pin-count significantly, and is a faster mechanism than conventional address decoders.

(5) Since a serial shift-path was already needed to support the global token mechanism, we elected to make multiple use of this path. During system initialization, this inter-chip path can be diverted on each chip into the 'configuration' register that programs the supertree, thus linking all configuration registers into one large scan-path.

(6) Reasonable yield from fabrication at 4 micron feature size allows only 64 pixels on a chip (1.5 micron feature size would allow a few hundred pixels per chip). Current fabrication limits led us to investigate other ways of getting more memory on a single packaged device. We saw that in future chip implementations, the 64-pixel chip might become merely one of a number of modules on a much larger chip, where some modules are allowed to be faulty. An Alive register was installed on the chip to provide a way of turning off faulty modules under software control. On initialization, these registers can, like the configuration registers, be linked together by the serial scan-path. A pattern of 1's and 0's scanned in corresponding to good and faulty modules. Modules (chips) with Alive set to 0 are disabled for video output, and their configuration registers are disconnected from the scan-path during supertree programming.

As in the *Pxpl2* prototype, a complete functional simulator was written for each of the image-generation functional blocks, the Translator, IGC, and Frame Buffer. This simulator could produce

crude images to help check the correct operation of various algorithms. The simulators for the Translator and IGC, with slight modifications, became the driver programs for the actual hardware.

Chip testing was done essentially on the display system itself. Since the memory chips are intended to produce graphics images, we simply plugged a single chip into the prototype display, exercised its functions, and observed the results on a color monitor where groups of memory bits were interpreted as color intensities. This rather crude testing strategy was surprisingly effective, even in diagnosing design faults.

Testing revealed several problems with the design:

(1) Over-aggressive use of the newly-available buried contacts in the memory (design rules for burieds were still rather vague) was most likely responsible for rather poor yield (approximately 20%).

(2) The dual-ported memory cell design was flawed and failed to decouple the two ports fully. Image-generation and video scan clocks therefore had to be synchronized.

(3) Failure to carry through a rigorous timing analysis of the memory system and its video output circuitry led to a timing fault that drastically reduced scan-out speed (approximately 1 MHz), but still allowed the chip to function. Under this limitation, the prototype display could be populated only with eight chips per logical board.

The system works correctly under restrictions imposed by the design flaws. Its speed is limited not by hardware design problems, but by the software that emulates the Translator and IGC. Since this software runs about 1000 times slower than the on-chip processors, the system is fast enough to produce only very crude animation (about 2-3 updates per second on an image with 6 polygons).

The module-level fault tolerance scheme using the Alive register was successfully tested on the *Pxpl3* prototype. In fact, the entire serial-shift mechanism for Alive, supertree configuration, and global-token passing worked successfully on first silicon.

Building and testing the *Pxpl3* prototype brought forcefully to our attention the need to build hardware to execute the Translator and IGC functions. The experience also suggested three important design changes in the memory chip:

(1) The fabrication yield for the *Pxpl3* chips would have been greatly improved (better than 2x) with the addition of a redundant memory column and a redundant row.

(2) The dual-ported memory scheme did not appear to be a very effective way to support scan refresh, even had it been successfully implemented. It provides much higher bandwidth in the second port than is required by the scan-out process and requires a memory cell about twice the size of a conventional cell.

(3) Since the multiplier tree in *Pxpl3* is implemented essentially as shown in Figure 4, the tree must be flushed after the formation of each result, in order to clear the carry registers at each node. A 30% speedup could be achieved if the multiplier were more fully pipelined.

3.3 Pxpl4

The improvements suggested by the *Pxpl3* prototype have been built into a new enhanced memory chip (*Pxpl4*), in fabrication at the time of writing. The chip contains 64 pixels, each with 72 bits of memory. In 4-micron nMOS, active circuitry (excluding pad frame and wiring) is 7.5 x 4.0 mm and contains about 33,000 transistors. Of this area, about 70% is devoted to memory, 20% to the binary-tree circuitry, and 10% to the pixel ALU. With MOSIS's 3-micron fabrication, two modules (128 pixels) can be built inside a MOSIS-standard pad frame.

The system built around this chip will be expandable to 512 x 512 pixels with 72 bits/pixel (or it can display 1024 x 1024 pixels with 18 bits/pixel). This system will be hosted by a high-performance workstation that will store and manipulate image data-bases, provide user interaction, and initially carry out part of the polygon transformation tasks in scene generation. (Later versions of the

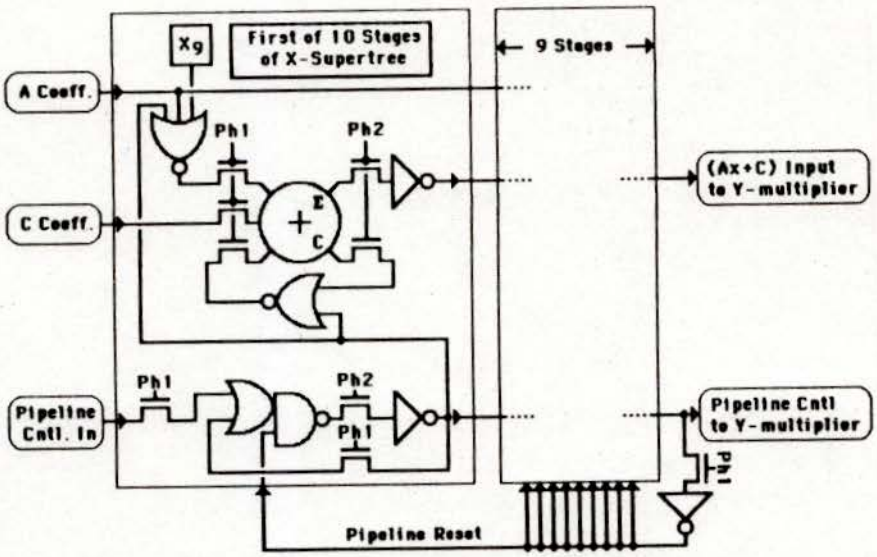

Figure 8: Circuitry for pipelining tree operations. The X-supertree is shown, but the scheme is used in the Y-tree as well.

system will perform transformations using special hardware, such as the Geometry Engine in the Silicon Graphics IRIS [Clark, 1982]).

The following paragraphs detail various design enhancements in the current memory chip.

Multiplier Pipelining

Multiplier operations are fully overlapped by including a small amount of additional hardware for a pipeline register. Figure 8 shows the details of this scheme, which differs somewhat from that in conventional pipelined multipliers [Lyon, 1976]. The pipelining register is 'sticky': when it receives a logic-1 it is locked into this state until a global clear is generated. Thus, a stream of 1's marches down the multiplier just behind the formation of partial products contributing to the MSB of the result, and just ahead of the LSB of the new constant coefficient. When the stream of 1's reaches the last stage, a clear is generated that simultaneously re-enables multiplication at all stages.

Memory Design

Pxpl4 uses a 4-transistor dynamic memory cell that has the useful property of refreshing itself during a read operation. Since each memory row is connected only to its pixel ALU, no special sense amplifier is needed for read access; simulations show that the memory operates faster (about 20 MHz) without one.

The video output port of the pixel memory is implemented as a single double-buffered register per chip, the Shadow Register, in which a copy of the currently selected pixel's memory is built up sequentially. The scheme is shown in Figure 9.

A pixel selector points at the pixel (memory row) needed for the next scan-line and puts a copy of the data from each bit onto a one-bit bus during each read or write cycle. Simultaneously, the memory address decoder output is delayed and used to load data from this bus into the element of the shadow register corresponding to the selected memory bit. Thus, as each bit of memory is 'visited' during image generation, it is copied into the master half of the Shadow Register. At the end of a scan-line, the Video Controller unloads the master into the slave of the Shadow Register, where the data is available for output. The Shadow Register mechanism is much more space-efficient than a full dual-ported memory. It requires some care, however, in design and in operation to avoid data corruption due to synchronization failure and to ensure that image-intensity bits are visited

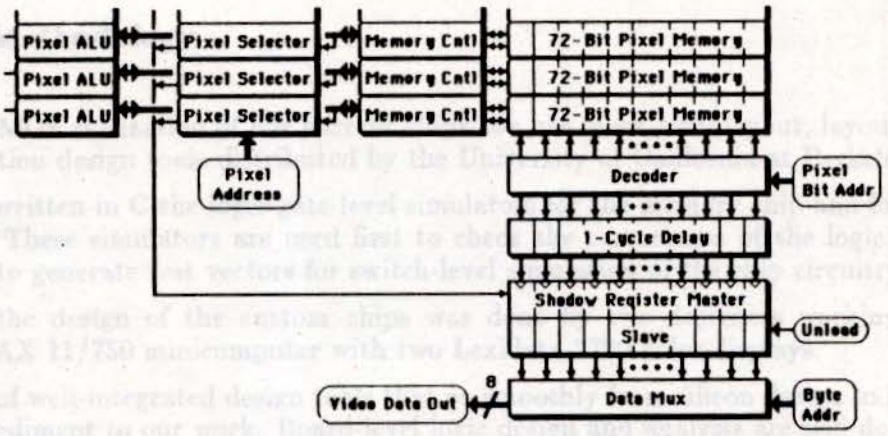

Figure 9: Video memory port implemented as a Shadow Register.

often enough to update the register once per scan-line. Neither of these problems is difficult to overcome in practice.

Redundant Modules and Circuits

Pxpl4 retains the Alive mechanism for module fault tolerance that was tested in *Pxpl3* and adds circuitry to support redundant memory elements to make each module more robust.

The chip contains one extra memory column. A redundant-column address register is added to the chip's configuration register, so that the address of the column to be replaced can be scanned in during system initialization.

Provision of a redundant row is somewhat more difficult, since one of the ALU-memory interfaces must be re-connected to the redundant memory row. Re-connection cannot be readily implemented without undue loss in system speed, so instead we provide an entire extra pixel with ALU and complete sub-tree path. The 6 nodes of local tree above the redundant pixel are realized simply by building a full 20 stages of supertree. The configuration (address) registers in these stages contain the address of the redundant pixel, and are loaded with the rest of the address at initialization. Redundant row and column enables are also provided to turn the entire mechanism off.

The redundant column circuitry requires only about 1.4% of the total active circuit area and the redundant row about 5.3%.

3.4 Buffered Pixel-Planes

One drawback of our present system is that the full parallelism cannot be utilized subsequent to scan-conversion. During visibility and painting calculations, all pixels outside the currently processed primitive are idle.

We are investigating an alternative system design, called 'Buffered Pixel-Planes', that improves parallelism. A modified Image Generation Controller with accept/reject circuitry and a FIFO is fully integrated onto a custom chip, and many copies are distributed across the system, each supervising a group of enhanced memory chips. The Translator sends bounding-box data for each primitive ahead of its coefficients and instructions. Each IGC accepts or rejects the current primitive based on the bounding box; if inside, coefficients and instructions are accepted and pushed into the FIFO for processing.

We have simulated the behavior of such a system processing images of moderate complexity (up to 1000 polygons), and we predict approximately 5-fold speedup with modest (10-polygon) FIFO size.

4. Design Methodology

4.1 Tools

For the nMOS realization of our current chips, we use mask-level layout, layout analysis, and circuit simulation design tools distributed by the University of California at Berkeley.

We have written in C the logic-gate-level simulators for the memory chip and for other system components. These simulators are used first to check the correctness of the logic design for the system, then to generate test vectors for switch-level simulation of the chip circuitry.

Most of the design of the custom chips was done by two designers working on a Digital Equipment VAX 11/750 minicomputer with two Lexidata 3700 color displays.

The lack of well-integrated design tools that go smoothly from silicon design to board design is a serious impediment to our work. Board-level logic design and analysis are still done using paper and pencil, with considerable assistance from standard UNIX program-development tools. Boards are layed out with a graphic chip-layout editor and fabricated using MOSIS's PC-board service.

For some time we have been working on a CMOS version of the enhanced memory chip. Mask-level design of CMOS projects is unattractive for two reasons: First, the additional complexity of CMOS technology makes an already-difficult layout task much more tedious. Second, the fabrication technology is developing rapidly, and it is not clear that scalable design rules for mask layout will be an effective way of tracking these advances. We have therefore been using (and assisting in the development of) the VIVID* symbolic-layout design system [Rosenberg *et al.*, 1985]. The system includes a hierarchical layout compactor that translates symbolic layout to mask with the help of a technology file that captures all relevant information about a particular fabrication process. In this way a given symbolic design can hope to survive considerable change in the target fabrication technology.

4.2 Design Style

Constructing a full-scale, full-speed system is a much more complex task than building a small prototype. The principal lesson learned from our early prototype construction was the need for complete documentation and precise interface specifications. We have therefore adopted for all system components a design style whose elements are:

(1) The system is decomposed into modules following a restricted hierarchy, in which only leaf-cells are allowed to contain circuit elements. The hierarchy is maintained in parallel in the physical domain (*e.g.*, chip layout) the logical domain (*e.g.*, logic schematics), and the behavioral domain (*e.g.*, simulators that model the logic). Composition cells may contain only interconnection information (abuttment, for example, within a chip layout) and other cells. Leaf cells may contain only circuit elements (logic gates in the logical description; transistors, wires, contact cuts in the physical description).

(2) Borrowing from strongly-typed programming languages, we impose a strong-typing scheme on all signals in the system. To ensure that modules are 'correctly connected' (*e.g.*, timing conventions and active-levels are observed), only a few signal types are allowed for connection between modules. The typing scheme is based on non-overlapping multi-phase clocks, and if applied carefully, avoids race conditions in sequential circuits. The signal types are encoded in a suffix attached to every signal name, providing a powerful documentation aid.

(3) Special hazards are involved where clocking convention (*e.g.* edge-triggered vs. level-sensitive latching) and implementation technology (TTL logic vs. custom MOS) changes, particularly at the chip I/O pads. To help assure that this interface will work properly, we define its timing conventions in a simple way, using two-sided timing constraints.

(4) Every major module in the design is modeled by a functional simulator. The simulated modules are tested separately, then plugged together to check the correctness of interfaces and overall

* VIVID is a trademark of the Microelectronics Center of North Carolina

operation of the simulated system. The simulators provide test vectors for chip simulation and testing.

The signal-typing/timing schemes are similar to [Noice *et al.*, 1982] and [Karplus, 1984]. Other elements of the style were influenced by [Lattin *et al.*, 1981; Stefik and Conway, 1982; Stefik *et al.*, 1982], among other sources.

4.3 Clocking Techniques

Our nMOS custom chips use a high-voltage clocking scheme ('hot clocks') suggested to us by [Seitz, 1982] and described in [Seitz *et al.*, 1985]. The main advantage of the technique is that n-enhancement transistors transmit a logic-HI without threshold drop and at much higher speed. In general, this clocking method produces layouts that are denser and much faster than conventional single-supply designs.

In a system with many custom chips, it is extremely inconvenient to generate these clock signals off-chip at a non-standard voltage. We have therefore built on-chip clock drivers that perform level translation and single-to-2-phase conversion. A separate input pin, biased typically at 8 volts, powers only these circuits. We have successfully built and tested a number of such high-voltage drivers, and our current design charges 100 pf to 7 volts in about 10 nsec.

The clock signals are produced in a single generator on the chip and distributed, so far as possible, continuously in metal wiring. For routing purposes, the clocks are second in importance only to Vdd and ground. For the inevitable cross-unders, we use 'low resistance wire', essentially an extended buried contact whose sheet resistance we have measured at about 7-8 ohms/square.

Level-sensitive register controls require qualified clocks that are generated in clocked, bootstrapped drivers [Joynson *et al.*, 1972]. The design of these compact drivers is not difficult, and they can be made to generate qualified clocks that follow the primary clock signal with nearly zero delay.

5. Pixel-Planes Algorithms

In this section we briefly describes how polygonal images are processed in *Pixel-planes*, and we outline several new algorithms, more fully described in [Fuchs *et al.*, 1985]. The timing estimates in this section assume that the *Pxpl4* chips are clocked at 10 MHz.

Rendering smooth-shaded polygons requires scan conversion, hidden surface elimination, and shading calculations:

Scan conversion is outlined in Figure 10. Processing begins by enabling all pixels in the display. Edges are encoded in linear equations of the form $F(x,y) = Ax + By + C = 0$, and the sign of F is tested at every pixel to determine visibility. Scan conversion leaves pixels outside the current polygon disabled; only those inside participate in further visibility and shading calculations.

Hidden surface elimination can be performed using a depth-buffer algorithm in which the z-coordinate of a pixel is encoded in a set of coefficients A, B, C by the linear expression $z = Ax + By + C$. Each pixel stores a value of z for the closest polygon so far processed and compares this value with the incoming z. If the new z is closer, the current polygon is visible at this pixel, and it remains enabled for shading, updating its z-buffer. If the stored z is closer than the new z, the pixel is disabled during shading.

Smooth shading is accomplished by computing a set of coefficients for each of R, G, and B, so that the color-intensity at each pixel is approximated by $F(x,y)$. Gouraud-like smooth shading can be carried out by painting each multi-sided polygon as a series of triangles (scan-conversion and hidden surface elimination, however, need only be done once for each polygon).

Polygon processing time depends on the number of edges and the number of bits needed to represent the function $F(x,y)$ for each operation. Approximately 30,000 4-sided polygons of arbitrary shape and orientation can be processed per second, using the steps outlined above.

Polygon input data:
A_i, B_i, C_i **for each edge** i

For each edge i **define:**
$F(x,y) = A_i x + B_i y + C_i$

Pixel at (x,y) **is inside polygon if and only if:**
$F_i(x,y) > 0$ **for all** i

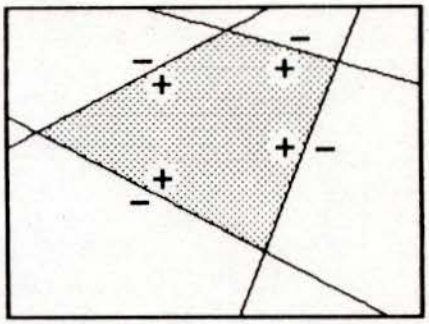

Figure 10: Scan-coverting polygons using linear expressions.

Shadows are important depth cues in interactive systems, and we have developed a method, similar to [Brotman and Badler, 1984], for casting shadows from arbitrary light sources using using shadow volumes [Crow, 1977]. For each polygon in the image, the set of visible pixels that lie in the frustum of the polygon's cast shadow are determined, and the color intensity of these pixels is diminished by an appropriate factor. Shadows are post-processed after a non-shadowed polygon image has been generated. The shadows for approximately 78,000 polygons can be computed per second.

Filled circles can be rendered rapidly in *Pixel-planes* by treating a circle as a polygon with one edge. The method separates the equation of a circle into a linear part that differs for each circle size and position, and a quadratic part that is the same for all circles. The quadratic part is pre-computed and its distinct values are loaded into every pixel at system initialization. Circles are processed by encoding center-position and radius in coefficients A, B, C and adding the linear expression to the stored quadratic term at each pixel. This method can readily be extended to render the other conic sections, such as ellipses. Spheres can be approximated by a quadratic surface, depth-sorted using a Z-buffer, and highlighted from an arbitrary light source. Approximately 34,000 spheres can be processed per second.

Texture mapping can be performed by using the linear expression evaluator to compute a texture plane address at every pixel. The appropriate color value for a pixel is then looked up in a texture table, transmitted entry by entry to the Smart Frame Buffer.

Anti-aliasing may be accomplished by one of two methods. The first, similar to 'super-sampling', blends a newly computed image with a previously computed image in a series of steps that successively refine the image. To support rapid interaction, the image is only refined when stationary. A second approach uses a method similar to that used on the Evans and Sutherland CT-5 real-time image generation system [Schumacker, 1980]. This method assumes that a visibility ordering of the polygons has already taken place, and uses a sub-pixel coverage mask to compute the anti-aliased image.

Transparency effects can be produced using the sub-pixel coverage mask for successive refinement, or by disabling patterns of pixels (*e.g.* a checkerboard) during polygon processing.

Adaptive Histogram Equalization (AHE) [Pizer *et al.*, 1984] is a powerful image processing technique used for grey level assignment and contrast enhancement of Computed Tomographic (CT) images. A local histogram is computed for every pixel in the image, and then used to compute a new grey level assignment for that pixel. For a 512 x 512 image, this method requires about 5 minutes of computation on a VAX 11/780, and is therefore too inefficient for most uses. The parallel processing power of *Pixel-planes* can be used to compute simultaneously the grey level assignment for each pixel in the image, without the need for histogram calculation. A rank counter, maintained in a portion of each pixel's memory, can be incremented using the pixel ALU. The intensity of a given pixel is broadcast and compared, in parallel, to the intensity of all pixels

that are within a local region. The rank counter is incremented at all pixels in the local region whose intensity is greater than the given pixel. After all pixels have been processed, the rank counter values are scaled and displayed. We estimate a 512 x 512 image will require approximately 4 seconds to compute on *Pixel-planes*.

6. Comparison with Other Architectures

We divide alternative VLSI-based architectures for graphics into two classes (as outlined in [Abram and Fuchs, 1984]): those that divide the image plane into sub-planes, with a processor for each subdivision, and those that divide the object database, assigning a processor to each subdivision. The *Pixel-planes* system is an example of the former, and we therefore compare it with two other systems of this type.

6.1 Architectures for Image-plane Subdivision

Several groups ([Fuchs and Johnson, 1979]; [Clark and Hannah, 1980]; [Gupta *et al.*, 1981]) have proposed systems that make more effective use of commercial RAM chips than conventional frame buffers; we refer to this as the *interlaced* approach. In [Clark and Hannah, 1980], the RAM's are interlaced so that on any 8 x 8 area of the screen, one pixel comes from each of the RAM's; each memory contains every eighth pixel in every eighth row. The scheme uses two layers of special processors organized in columns and rows, with a row-processor in charge of each RAM chip (or group of RAM chips when more than 1 bit/pixel). An entire 8 x 8 patch on the screen can be accessed with a single memory reference by the 64 row processors, so a polygon (or other primitive) roughly the size of a patch, or larger, can be processed with considerable parallelism.

A major advantage of the *interlaced* approach is that it uses high-density commercial RAMs and yet achieves performance greatly improved over conventional frame buffers with relatively few custom chips. This design is hampered, however, by the bandwidth limitations imposed by separating memories and processors onto separate chips.

Another recent approach, described in [Demetrescu, 1985], employs 'Scan-Line Addressable Memories' (SLAM's). A system with 1024 x 1024 one-bit pixels is organized in 16 rows, each with a Scan-Line Processor in charge of 4 SLAM chips. Each of these units contains and controls all of the pixels in 64 successive scan-lines. Each SLAM chip contains a conventional RAM array, organized as 64 rows of 256 one-bit pixels, augmented with an array of very simple processors that operate in parallel on all pixels in a row. In one cycle of operation, all pixels in 16 scan lines can be accessed. The Scan-Line Processors provide buffering of graphics primitives, so that very high parallelism can be achieved.

The system-level implementation of a SLAM-based display should be very clean. In contrast to the *interlaced* design, high-bandwidth memory-processor communication wiring is completely encapsulated in the SLAM chips. Commands and data are broadcast from each Scan-Line Processor to its SLAM's over a low-bandwidth bus. The SLAM design solves the display-refresh problem without interrupting image processing (by including a display shift register on the SLAM chip). These are the principal features common to the SLAM and *Pixel-planes* approaches.

6.2 Comparison with Pixel-planes

For today's high-performance workstations, where the display requires one or a few bit-planes and handles (mainly) multiple windows with text, lines, and flat-shaded polygons, the SLAM approach is extremely attractive. For such applications, it appears to be considerably faster than either the *interlaced* or *Pixel-planes* designs and is several orders of magnitude faster than conventional frame buffers. The cost of the approach, like ours, is the need to use custom-designed memory chips. The processors on the SLAM chip are extremely simple and appear to require very little area, however, perhaps as little as 1/10 that of our processors.

The *Pixel-planes* system is targeted at applications more demanding than the displays in current workstations, such as medical display and imaging, molecular modeling, mechanical design systems,

and flight and navigational simulators. These applications require interaction with 3D images needing visibility determination, smooth shading, shadows, and textures; images with perhaps thousands of primitives and significant depth complexity must be updated at frame rates.

Methods for improving perceived image quality necessarily rely on storing additional information at each pixel. Clearly, the most effective means of improving performance is accessing and processing this data in parallel, closely associating a large amount of pixel memory with a pixel processor. The *Pixel-planes* design provides the power of a processor per pixel, at relatively modest cost in silicon area, and a very general method for computing images.

The *interlaced* approach cannot grow gracefully in the dimension of bits/pixel because of chip I/O limitations. In the SLAM design, one alternative for growth in this direction is multiple banks of SLAM, one for each bit plane. To expand such a system to the size accommodated by the current *Pixel-planes* chip would entail a copy of the processor for each of 72 bit-planes, an intolerable increase in silicon area. The other alternative is using a column in the SLAM to hold all bits of a pixel, then bit-serially processing data; this alternative is similar to our approach, but it fails to provide a very general image-computation method.

For applications that require accessing large amounts of memory per pixel, our system should be denser and faster than either of the other approaches. In effect, we have already paid the price of accessing many bits/pixel: bit-serial data access and a more general (and costly) method of display refresh.

7. Acknowledgements

We wish to thank Vernon Chi (Director), Mark Monger, and John Thomas, of the UNC Microelectronic Systems Laboratory, for design and technical assistance in building the *Pixel-planes* system. We also wish to thank Alan Paeth and Alan Bell of Xerox Palo Alto Research Center for collaborating in the design of *Pxpl2* and *Pxpl3*, Scott Hennes for assistance with the *Pxpl3* chip, Fred Brooks for the basic circle scan-conversion algorithm, and Turner Whitted for discussions about anti-aliasing and transparency algorithms.

8. References

Abram, G. D. and H. Fuchs. July, 1984. "VLSI Architectures for Computer Graphics," *Proceedings of the NATO Advanced Study Institute on Microelectronics of VLSI Computers*, Sogesta-Urbino, Italy.

Brotman, L. S. and N. I. Badler. October, 1984. "Generating Soft Shadows with a Depth Buffer Algorithm," *IEEE Computer Graphics and Applications*, 5–12.

Clark, J. H. and M. R. Hannah. 4th Quarter, 1980. "Distributed Processing in a High-Performance Smart Image Memory," *LAMBDA*, 40-45. (LAMBDA is now VLSI Design).

Clark, J. H. July, 1982. "The Geometry Engine: A VLSI Geometry System for Graphics," *Computer Graphics*, **16**(3), 127–133. (Proc. Siggraph '82).

Crow, F. C. July, 1977. "Shadow Algorithms for Computer Graphics," *Computer Graphics* , **11**(2), 242–248. (Proc. Siggraph '77).

Demetrescu, S. May, 1985. "High Speed Image Rasterization Using Scan Line Access Memories," *In these Proceedings.*

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

Fuchs, H. and J. Poulton. 3rd Quarter, 1981. "Pixel-planes: A VLSI-Oriented Design for a Raster Graphics Engine," *VLSI Design*, **2**(3), 20–28.

Fuchs, H., J. Poulton, A. Paeth, and A. Bell. January, 1982. "Developing Pixel Planes, A Smart Memory-Based Raster Graphics System," *Proceedings of the 1982 MIT Conference on Advanced Research in VLSI*, Dedham, MA, Artech House, 137–146.

Fuchs, H., J. Goldfeather, J. P. Hultquist, S. Spach, J. D. Austin, J. G. Eyles, and J. Poulton. January, 1985. Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes, Technical Report 85–002, Dept. of Computer Science, University of North Carolina at Chapel Hill.

Gupta, S., R. F. Sproull, and I. E. Sutherland. August, 1981. "A VLSI Architecture for Updating Raster Scan Displays," *Computer Graphics*, **15**(3), 71–78. (Proc. Siggraph '81).

Joynson, R. E., J. L. Mundy, J. F. Burgess, and C. Neugebauer. June, 1972. "Eliminating Threshold Losses in MOS Circuits by Bootstrapping Using Varactor Coupling," *IEEE Journal of Solid-State Circuits*, **SC-7**, 217–224.

Karplus, K. August, 1984. A Formal Model for MOS Clocking Disciplines, Technical Report 84-632, Dept. of Computer Science, Cornell University, Ithaca, NY.

Lattin, W. W., J. A. Bayliss, D. L. Budde, J. R. Rattner, and W. S. Richardson. 2nd Quarter, 1981. "A Methodology for VLSI Chip Design," *LAMBDA*, **2**(2), 34–44. (LAMBDA is now VLSI Design).

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

Noice, D., R. Mathews, and J. Newkirk. 1982. "A Clocking Discipline for Two-Phase Digital Systems," *Proc., IEEE International Conference on Circuits and Computers*, 108–111.

Pizer, S. M., J. B. Zimmerman, and E. V. Staab. 1984. "Adaptive Grey Level Assignment in CT Scan Display," *Journal of Computer Assisted Tomography*, **8**(2), 300–305.

Rosenberg, J. B., C. D. Rogers, and S. Daniel. 1985. "An Overview of VIVID, MCNC's Vertically Integrated Symbolic Design System," *To appear in the Proceedings of the 1985 Design Automation Conference.*

Schumacker, R. A. November 1980. "A New Visual System Architecture," *Proceedings of the 2nd Annual IITEC*, Salt Lake City.

Seitz, C. 1982.Private Communication.

Seitz, C. L., A. H. Frey, S. Mattisson, S. D. Rabin, D. A. Speck, and J. L. A. Snepscheut. May, 1985. "Hot-Clock nMOS," *In these Proceedings.*

Stefik, M., D. Bobrow, A. Bell, H. Brown, L. Conway, and C. Tong. January, 1982. "The Partitioning of Concerns in Digital System Design," *Proceedings of the 1982 MIT Conference on Advanced Research in VLSI*, Dedham, MA, Artech House, 43–52.

Stefik, M. and L. Conway. April 28, 1982. Toward the Principled Engineering of Knowledge, KB-VLSI-82-18, Xerox, Palo Alto.