To appear in Advances in Graphics Hardware 2: Proceedings of the Eurographics '87 Second Workshop on Graphics Hardware

PIXEL-PLANES 4: A SUMMARY

John Eyles, John Austin*, Henry Fuchs, Trey Greer, John Poulton

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

Abstract

We describe the current state of the Pixel-planes research project, whose goal is to develop powerful raster graphics systems for the next generation of workstations. The first full-scale prototype has been in regular use in our department's computer graphics laboratory since its first demonstration at SIGGRAPH '86, more than a year ago. We describe the final hardware configuration of the prototype system, filling in some of the engineering details heretofore unpublished. Next we outline the programming environment for the machine and summarize the major algorithms that have been developed. Finally we discuss our progress towards a new generation of the Pixel-planes architecture, Pixel-planes 5.

1. Motivation and Overview

The goal of the Pixel-planes research project has been to build an affordable and generally useful system for interactive 3-D display. As personal dedicated computers have matured over the years, an increasing percentage of machine resources has been devoted to the user interface. The idea of giving each user a dedicated processor and high resolution display, although once controversial, has now been widely adopted and forms the basis for modern desk-top workstations [Kay, 1977; Thacker et al., 1979]. The next plateau in workstation performance will be reached, we believe, when desk-top systems allow the user to manipulate and modify realistic, 3-D, colored images of objects in real time. The availability of such systems at a reasonable price should have a great impact in such diverse areas as mechanical design, medical diagnosis and therapy, molecular modelling, architectural design, and vehicle simulation for pilot and driver training. Workstations with real-time 3-D displays will likely be characterized by hardware largely devoted to the user interface, because the computational task of rendering realistic images of objects is extremely demanding, far beyond the capabilities of the most powerful existing general-purpose computers.

* Now with Sun Microsystems, Inc., 500-C Uwharrie Ct., Raleigh, NC 27806

Pixel-planes is a raster graphics system for rapidly rendering 3-D objects and scenes. It features a 'smart' frame buffer composed of custom, logic-enhanced memory chips that can be programmed to perform most pixel-oriented tasks in parallel at each pixel. The novel feature of the approach is a unifying mathematical formulation that expresses the parallel execution of these tasks. This formulation is supported on an efficient tree-structured computation unit that calculates inside each chip the proper values for every pixel in parallel.

The 'front-end' of the system specifies the objects on the screen in a pixel-independent form and broadcasts this description to the custom memory chips that form the frame buffer. These chips generate an image directly from this description. Image primitives such as lines, polygons, and spheres are each described by expressions (and operations) that are linear in screen space, that is by coefficients A,B,C such that the value needed at a pixel is Ax+By+C, where *x,y* is the pixel's location on the screen. These ABC's and operation codes are passed to the frame buffer and processed simultaneously at every pixel.

Pixel-planes is a radical approach to raster graphics [Fuchs and Poulton, 1981; Fuchs et al., 1982; Poulton et al., 1985; Fuchs et al., 1985]. The information passed to the frame buffer is not address/data pairs (*x,y* addresses, RGB data) as in conventional systems, but linear expressions processed simultaneously at every pixel. While other raster graphics systems perform the most time-consuming calculations either on general-purpose processors or on special hardware that executes only a particular set of graphics functions, Pixel-planes is a general-purpose engine, especially powerful when the pixel operations can be described in terms of linear expressions.

2. Hardware Configuration

The Pixel-planes 4 system (Photograph 1, Figure 1) consists of a Digital Equipment Corporation MicroVax II workstation, which acts as host, and a separate cabinet which contains the prototype custom hardware; the prototype contains two racks: a Multibus and a fully custom backplane. The host is connected to the prototype by a DMA link. The DMA link is supported at the host end by a DR11W card and in the Pixel-planes prototype by a custom 'host interface' in the Multibus. The host controls the system by performing reads and writes on the Multibus using this link.

The prototype is composed of two parts: a Graphics Processor and a Frame Buffer. The Graphics Processor is a floating-point uni-processor; it traverses a hierarchical display list, computes viewing transformations, performs lighting calculations, clips primitives that are not visible, performs perspective division, and translates the resulting screen-space descriptions into the form of data (A,B,C) for linear expressions, together with instructions. The Frame Buffer is made from custom VLSI processor-enhanced memories and is 512 x 512 pixels by 72 bits per pixel in size; it accepts the word-parallel A,B,C coefficient sets and instruction op-codes to create raster images and refreshes a standard RGB monitor.

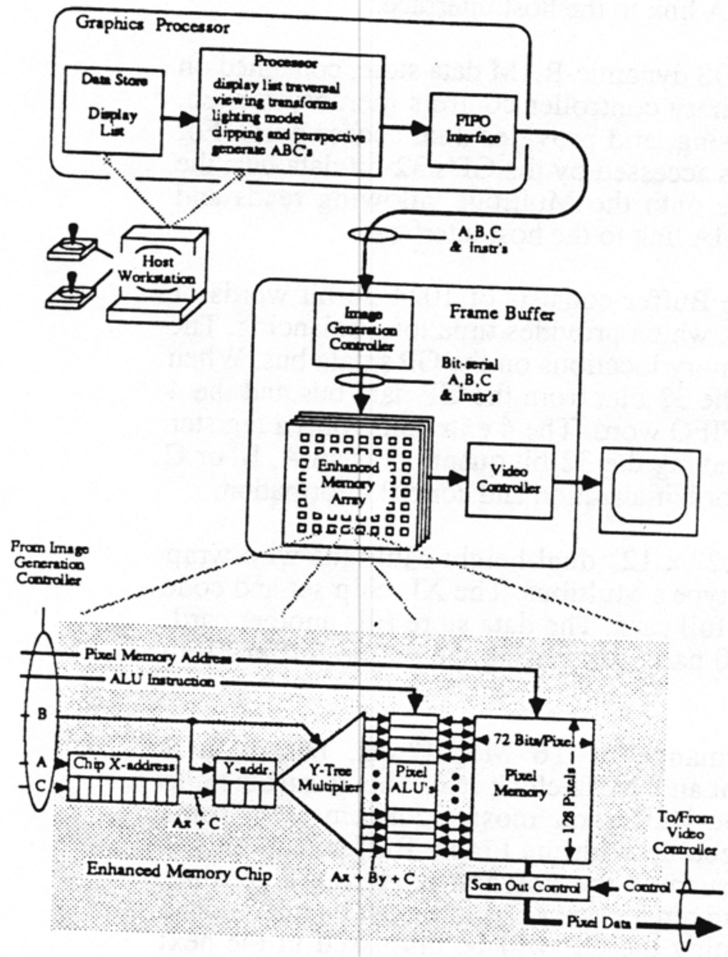


Figure 1: System block diagram; exploded view shows details of the custom processor-enhanced memory chip, an array of which forms the 'smart' frame buffer.

2.1 Graphics Processor

The Graphics Processor (GP) is implemented using the Weitek 8032 XL chip set. The chip set consists of a sequencer chip, an integer ALU chip, primarily used to compute instruction addresses, and a floating-point ALU chip. The GP is a Harvard architecture, with separate busses and memory systems for instructions and data.

The static-RAM code store has 512 KBytes, arranged in 64 bit words. Instructions are fetched using the GP's code address and code busses. Although the Weitek architecture supports instruction caching, we have not implemented it in this system. The code store is read-only from the GP's code bus, and write-only from the Multibus; this allows easy loading of code from the host, using the DMA link to the host interface.

The GP has 8 MegaBytes of CMOS dynamic-RAM data store, contained on 1 MegaBit parts. A custom memory controller controls memory refresh, supports fast page mode addressing, and provides dual access ports. For normal operation the data store is accessed by the GP's 32-bit data bus; the second port maps the data store onto the Multibus, allowing reads and writes from the host using the DMA link to the host interface.

The GP's interface to the Frame Buffer consists of 1024 36-bit words of first-in/first-out (FIFO) memory, which provides time load balancing. The interface is mapped onto 16 memory locations on the GP's data bus. When the GP writes to the interface, the 32 bits from the GP data bus and the 4 LSB's of address form a 36-bit FIFO word. The 4 extra bits form a register address for the Frame Buffer, and tag the 32-bit quantity as an A, B, or C coefficient, instruction op-code, or initialization and control information.

The GP is contained on three 12" x 12" dual-height Multibus wire-wrap cards, which reside in the prototype's Multibus. The XL chip set and code store are contained on one half-full card. The data store fills another card. The third card contains about 20 parts comprising the Frame Buffer FIFO interface.

The GP attains peak performance of 16 MegaFlops, based on 1 multiply-accumulate per cycle at an 8 MHz clock rate. This is sufficient to match the speed of the Frame Buffer on most algorithms. On some algorithms the GP is able to keep up with the Frame Buffer with only *C* coding. On other algorithms, such as the polygon rendering algorithm, inner loops (i.e.: the procedure to render one polygon) must be microcoded in order to keep up. Programming the GP will be discussed in the next section.

2.2 Frame Buffer

The Frame Buffer consists of 3 parts: the array of custom VLSI processor-enhanced memory chips, the Image Generation Controller, and the Video Controller.

The custom chips that compose the frame buffer have three main parts: a conventional memory array that stores all pixel data for a 128-pixel column (72 bits per pixel), an array of 128 tiny one-bit ALU's that carry out pixel-local arithmetic and logical operations, and a multiplier tree that generates bilinear expressions Ax+By+C simultaneously for all pixels. In generating these bilinear expressions, the multiplier provides the power of two 10-bit bit-serial M/A's at every pixel, but at much less expense in silicon area. The chips are implemented in 3-micron nMOS technology and contain approximately 63,000 transistors each. Of the chip's active area, 70% is dedicated to memory and 30% to processing circuitry.

An array of 2048 of these processor-enhanced memory chips forms the core of a massively parallel (512 x 512), bit-serial, single instruction, multiple data (SIMD) machine. On each micro-cycle, each pixel processor receives one bit of the tree result Ax+By+C, generated in the multiplier tree from global ABC bit-streams; the pixel ALU's execute a micro-instruction, and one bit in pixel memory is accessed. During any given micro-cycle, pixel ALU's for every pixel in the system execute the same micro-instruction simultaneously, and the same location in pixel memory is addressed at every pixel; however, although ABC are the same for every pixel, the tree result Ax+By+C will in general be different.

The enhanced memory chips are contained on thirty-two 15" x 15" printed circuit boards which reside in a custom backplane. The backplane provides (1) approximately 600 amps at 5 volts and 100 amps at 7 volts to power the memories, (2) a simple broadcast bus for distributing control signals to the memories, and (3) a daisy-chained path which links the boards together in a 'video train', which operates at pixel rates to collect video output data from the chips and channel it to the Video Controller. Each board contains 64 enhanced memory chips, 5 driver chips for distributing the control signals, and one stage of the 32-bit wide shift register which forms the video train.

The Image Generation Controller (IGC) and Video Controller (VC) together provide all control signals for the enhanced memory chips. The IGC provides the SIMD instruction for the pixel ALU's, the address into local pixel memory, and the bit streams for the A,B,C linear coefficients. The VC provides the control signals for the video output port of the enhanced memories and the video train, and contains color lookup tables and DAC's to generate analog RGB video. The IGC and VC function almost independently, with two exceptions. Firstly, the enhanced memories' video output ports require that the IGC address the 32 bits of pixel memory to be funneled into the video train at least once per video scan line (and, since the pixel memory is dynamic, it must address the remainder of the 72 pixel memory bits at least once every few milliseconds). Secondly, when doing a double-buffered display, the IGC must synchronize with a vertical retrace control signal from the VC, so that updating of the display buffer will occur during vertical retrace to avoid screen "tear".

The **Image Generation Controller** is based on a simple microcode sequencer, which generates the SIMD instructions for the array of enhanced memory, and a set of shift registers to bit serialize the A,B,C coefficients. The microcode sequencer has 256 words of code store, 32 bits in width. Sequence control is very simple: there is no stack; there are unconditional branches, and 5 types of conditional branches. For maximum efficiency and programmability the sequencer does not use pipelined instruction pre-fetch; each instruction is fetched during the first half of a clock cycle, and the instruction is executed and the next address generated during the latter half of the clock cycle.

Microcode is generated using a simple microcode assembler written in *C* which runs on the host. Microcode is loaded into the IGC's static-RAM microcode store via the FIFO interface from the GP. This allows easy rewrites and test of new IGC microcode.

The SIMD instruction for the pixel ALU's is generated directly from the microcode data register. The address into pixel memory is generated by a set of 3 counters: two are initialized from "destination address" and "source address" fields of the instruction op-code from the GP; the third generates addresses for memory refresh and to support the video scan-out mechanism, during cycles for which neither the destination address or source address is operative. The bit-streams for the A,B,C coefficients are generated by a set of shift registers which are controlled by microcode bits; the shifters also provide tokens representing the position of the LSB and sign-bit of the tree result Ax+By+C, which are used to qualify conditional branches, thereby synchronizing the ALU and pixel memory micro-instructions with the multiplier tree. This synchronization is necessary, since, for maximum efficiency, instructions can be overlapped; that is, while one instruction executes in the pixel ALU and pixel memory, the multiplier tree can generate the tree result for the next instruction.

The IGC is built using about 80 FAST-TTL parts, 8 high-speed MOS static-RAM's (for the microcode store), and one 8,000 transistor nMOS custom VLSI chip (for bit-serializing the coefficients). It is contained on one 12" x 12" Multibus wire-wrap card.

Video Controller. The VC also is based on a microcode sequencer. Bits of the microcode word define blanking and sync signals to support a variety of displays. Other fields control the video output port of the enhanced memory chips and the video train. The VC contains color lookup tables and D/A converters to produce analog RGB output. Like the IGC, the VC has a microcode assembler and a RAM microcode store accessible from the Multibus. The color lookup tables also are loaded from the Multibus. The VC is larger, more complicated, and more versatile than is necessary. It was implemented in this way in order to ease system development; for example, it was microcoded to fill the entire screen in an early prototype with only 64 x 64 pixels of enhanced memory. It also can be programmed to utilize the 72 bits of pixel-memory in different ways and in conjunction with various color lookup tables.

The VC is implemented using about 100 TTL and MOS memory parts. It is contained on one 12" x 12" Multibus wire-wrap card.

2.3 Input Devices

The Multibus rack contains a wire-wrap card containing a 16-channel analog to digital converter. A user input box contains two 3-degree of freedom, deflection-encoded joysticks and a slider potentiometer. These are connected to 7 of the A/D channels. The remaining channels are available for additional input devices. The A/D converter is accessed from the host MicroVax via the host interface and DMA link.

3. Software Environment

To program Pixel-planes, a programmer works with the three major components of the system: the host MicroVax workstation, the Graphics Processor, and the Frame Buffer. Typically, the host program downloads the executable for the GP and starts its execution, reads the user input devices and keyboard, and sends and receives messages and data to and from the GP program. The program on the GP does the more computation intensive algorithms to generate A,B,C coefficient sets and sends these coefficients along with instructions to the Frame Buffer.

The lowest-level programmer, who wishes to implement new algorithms for the Pixel-planes architecture, must be concerned with all three components of the system, including the details of rendering on the Frame Buffer. The following sections will be at this level, since it is most helpful for understanding the functioning of the system; however, some higher-level approaches are discussed in Section 3.8.

3.1 Frame Buffer programming model

The Frame Buffer is an array of 512 x 512 processors operating in SIMD fashion. These processors map directly onto the 512 x 512 display. Each pixel processor receives one output of the multiplier tree, Ax+By+C, where *x,y* is the address of the pixel processor on the display screen, and A,B, and C are the instruction coefficients; this *tree result* is produced in bit-serial form. Each pixel processor has 72 bits of local pixel memory and an ALU. This pixel-ALU operates bit-serially on data in the pixel memory, as well as on the tree result. The ALU also contains an Enable register, which qualifies all writes into pixel memory. On each microcycle, the multiplier tree receives a bit of the A,B,C coefficients, and each pixel processor receives a bit of the tree result Ax+By+C, an address in the 72 bit pixel memory, and an ALU micro-instruction. The pixel memory address, ALU microinstruction, and A,B,C coefficients are the same for all of the 512 x 512 processors. However, a different tree result is available at each processor since the tree result is dependent on the processor's *x,y* address; furthermore, execution of instructions at the pixel processors may be conditioned by the values of the individual Enable registers.

The bit-serialized A,B,C coefficients, ALU micro-instructions, and pixel memory addresses are generated by the IGC. The programmer deals only with the IGC and not directly with the pixel processors. The IGC effectively hides the bit-serial nature of the machine from the programmer. It receives 32- or 64-bit instruction op codes plus A,B,C coefficient sets as IEEE standard 32-bit format single-precision floating point numbers, and generates the cycle by cycle coefficient bits, ALU micro-instructions, and pixel memory addresses required to execute the specified instruction.

The instruction set for the IGC (that is, for the Frame Buffer) looks very similar to the instruction set of a very simple microprocessor; it supports addition, subtraction, comparison, and logical combinations of two quantities. However, the operand quantities, rather than being restricted to

8-, 16-, or 32-bit words, can be segments of pixel memory of arbitrary length; additionally, the tree result Ax+By+C can be an operand, so that an instruction can, for example, add the tree result to the contents of a segment of pixel memory.

Since the Frame Buffer is SIMD, branching is not be supported. Rather, a set of instructions is provided for modifying the contents of the ALU Enable register. For example, the Enable register can be loaded with, or logically combined with, a bit of pixel memory, the sign-bit of a tree result, or the result of a comparison of two pixel memory segments or the tree result and a pixel memory segment. Once a pattern of Enable register settings is created using these commands, all subsequent commands which perform write operations into pixel memory are effectively ignored by pixel processors with a 0 in their Enable register, since writes into pixel memory are disabled.

3.2 Interface from GP to Frame Buffer

The IGC is mapped onto several memory mapped locations in the GP's data space. (As described in Section 2.2, this interface is buffered with a FIFO). These memory locations can be thought of as an instruction op-code register and A, B, and C coefficient registers. Some instructions, which operate only on the data in local pixel memory and/or the Enable register, are invoked simply by writing to the instruction op-code register. For instructions which use the tree result, the programmer must first write the A,B,C coefficient values to those registers, and then write the op-code to the instruction register.

3.3 Programming the GP

A GP program accesses the Frame Buffer (IGC) registers using a set of macros with arguments, as supported by the *C* language pre-processor. These macros are of the form:

```
FB_A (coefficient value);
FB_B (coefficient value);
FB_C (coefficient value);
FB_INSTRUCTION-NAME (instruction arguments);
```

The 3 coefficient macros simply *push* the floating-point coefficient value to the correct memory address on the interface. The INSTRUCTION-NAME macro assembles the instruction arguments into a 32- or 64-bit IGC op-code and *push*-es this to the memory interface.

Code for the GP is written in the high-level language *C*. The *C* code is compiled on the host MicroVax using a cross-compiler supplied and supported by Weitek, and is downloaded to the GP via the DMA link. For applications in which the GP has difficulty keeping up with the Frame Buffer, portions of the GP code can be hand microcoded using an assembler, also supported on the MicroVax.

GP programs are linked with a library containing routines which handle details of loading microcode into the IGC and VC. Microcoding of the IGC and VC is beyond the scope of this discussion. A fairly complete set of IGC instructions and VC scan-out regimens has been created by the designers of those sub-systems. When required for a new application, additional microcode for the appropriate controller can be fairly easily generated; for example, writing and debugging IGC microcode for a new FB_ instruction generally requires about one hour.

3.4 Programming the host

Code for the host is written in *C* under Ultrix, DEC's version of UNIX. A library is provided for allowing the host program to read the user input devices via the A/D board. Another library handles communications with the GP via the DMA link; the routines allow the host program to download executables to the GP's code store, start and stop execution of GP programs, and transfer data to and from the GP's dual-access data memory.

Special assemblers on the host are used to generate microcode for the IGC, microcode for the VC, and configuration information for the enhanced memory chips (defining the *x,y* position of each chip's 128-pixel column on the display). Microcode for the IGC is downloaded to the GP and transferred to the IGC via its interface on the GP bus. Microcode for the VC is written to the VC's microcode store over the Multibus using the DMA link to the host interface. Enhanced memory chip configuration data is written to the VC via the Multibus and sent to the enhanced memory array using the scan-out control mechanism.

3.5 Communications between host and GP

Communications between the host and the GP can be done in two ways: using shared memory, and using interrupts.

Executables for the GP are generated using Weitek's assembler and linker, and the resulting executable is translated into an object module which is linked into the host executable. In doing this, the host program is permitted to access any variable declared as *external* in the GP program by prefixing the variable name with *gp_*; hardware support for this is provided by the GP's dual-access data memory. Using these variables, semaphores can be set up to allow the GP to signal the host that it has finished processing a frame, for the host to signal that another frame is ready, and so forth.

Alternatively, the GP can generate interrupts which are detected by the host interface, causing the DR11W to generate an interrupt on the host.

3.6 Video Controller support

The Video Controller supports both 8-bit and 24-bit modes of operation. In the 24-bit mode, the least significant 3 bytes of pixel memory (pixel memory addresses 0 through 23) are used to address the red, green, and blue color lookup tables; these lookup tables typically contain gamma corrections. The low-level programmer structures algorithms so that the desired red intensity for a pixel is placed in pixel-memory addresses 0 through 7, green at addresses 8 -15, and blue at addresses 16-23.

The 8 bit mode is used in more complicated applications when pixel-memory is too scarce to permit the "full color" 24-bit mode. In 8-bit mode, the red, green, and blue color lookup tables are all addressed by the least significant 8 bits of pixel memory (pixel memory addresses 0 through 7).The tables are loaded so that the 2 MSB's select white, red, green, or blue, and the 6 LSB's determine color intensity; no secondary or tertiary colors (other than white) are provided. The 8-bit mode can also be used with lookup tables that provide a monochromatic image, for image data such as medical ultrasound and CT scans.

3.7 Programming examples

A *C* program running on the GP might contain the following lines. This demonstrates rendering one triangle and updating the display buffer in a double-buffered application. In this example, the Z-buffer is 24 bits long, at pixel memory addresses 48-71, the image is computed using full 3-byte color at pixel memory addresses 24-47, and screen is refreshed from display buffers at pixel memory addresses 0-23.

```
.
.
.
FB_SETENABS ();
FB_A (..);  FB_B (..);  FB_C (..);  FB_TREEgeZERO ();
FB_A (..);  FB_B (..);  FB_C (..);  FB_TREEgeZERO ();
FB_A (..);  FB_B (..);  FB_C (..);  FB_TREEgeZERO ();
FB_A (..);  FB_B (..);  FB_C (..);  FB_MEMleTREE (48,24);
FB_A (..);  FB_B (..);  FB_C (..);  FB_LOAD (48,24);
FB_A (..);  FB_B (..);  FB_C (..);  FB_LOAD (24, 8);
FB_A (..);  FB_B (..);  FB_C (..);  FB_LOAD (32, 8);
FB_A (..);  FB_B (..);  FB_C (..);  FB_LOAD (40, 8);
.
.
.
FB_SETENABS ();
FB_CPY (0,24,24);
```

The commands SETENABS and TREEgeZERO are used to scan-convert the triangle. First, SETENABS sets the Enable register for all pixels. For the first edge of the triangle, the GP computes the equation of a line in the form $Ax+By+C = 0$, signed so that the tree result evaluates positive for pixels on the *inside* side of the edge and negative for pixels on the *outside* side of the edge. The GP sends this coefficient set to the Frame Buffer using the FB_A, FB_B, and FB_C macros, along with the TREEgeZERO ("tree greater than or equal to 0") instruction. This instruction causes the Enable register to be cleared if and only if $Ax+By+C$ is negative. TREEgeZERO instructions are sent for the other two edges. Finally, only those pixel processors for pixels inside the convex region of the triangle are left *enabled* (with 1's in the Enable register). (For larger convex polygons, additional TREEgeZERO commands are used for the additional edges).

To do Z-buffer hidden surface elimination, the GP computes the equation for the plane of the polygon in the form Z = Ax+By+C. This coefficient set is sent to the Frame Buffer with the MEMleTREE ("memory less than or equal to tree") instruction. This instruction compares the tree result to a segment of pixel-memory (specified according to its LSB and bit-length by the two instruction arguments), and clears the Enable register if the tree result is not less than or equal to the contents of the memory segment. This segment of pixel-memory, the Z-buffer, contains the Z-value of the "nearest" polygon which has been processed so far at that pixel; thus, those pixels which are hidden by previously drawn polygons are *disabled.*

The FB_LOAD instruction is then used to load the new values for the Z-buffer into pixel memory, for those pixels which are still *enabled* after scan conversion and hidden surface elimination. As with the MEMleTREE command, the two arguments to FB_LOAD specify the location and size of a segment of pixel memory used for the Z-buffer.

Finally, coefficient sets are for the red, green, and blue intensity planes are computed according to the Gouraud or other lighting model, and FB_LOAD is used to load the color intensity buffers in pixel memory.

After all triangles are rendered, the CPY command is used to transfer the contents of the red, green, and blue intensity buffers (at pixel memory address 24 through 47) to pixel memory addresses 0 through 23, which produce the RGB output when the VC is in 24-bit mode. Arguments to the CPY command are destination address, source address, and segment length.

Note that the locations and lengths of these segments of pixel memory are completely programmable, with the exception that the VC must be specially microcoded if pixel memory addresses other than 0-7 or 0-23 are to be displayed.

3.8 User and higher-level programming support

Higher-level programming. A fairly complete set of libraries has been written for host and GP programs. These are useful for the programmer who wishes to write a custom application, without being concerned with the details of rendering in the Frame Buffer described above.

For example, a user who wishes to define a custom user interface may only need to program the host. Library routines are used to set up display lists on the GP. The application then reads the input devices and perhaps some custom keyboard interface, and calls routines which cause frames to be generated from the display list according to a set of frame parameters which include viewing transformation matrices, lighting parameters, and the like.

A somewhat lower-level programmer may write code for both the host and the GP. However, rather than sending low level commands directly to the Frame Buffer, this programmer might call library routines on the GP which perform the beginning of frame setup in the Frame Buffer, then render polygons, spheres, and other primitives, and finally do the end of frame operations such as copying the image into the low 8 or 24 bits of pixel memory which refresh the display.

User support. We have developed standard programs for the user who wishes to interact with his/her databases at the high performance levels provided by Pixel-planes but who does not wish to do any programming.

For example, to use the program *front*, a user supplies a data base of polygon or sphere primitives, as an ASCII file in a generic format. *Front* downloads the data base to the GP via the DMA link, creating a display list in the GP's data store. Next *front* loads a program into the GP which interprets frame commands. *Front* then monitors the input devices interfaced to the A/D converter, and issues commands to the GP for displaying frames, using various viewing transformation matrices, lighting vectors, fields-of-view, and so forth. The GP program interprets these frame commands from the host, traversing the display list to generate an image in the Frame Buffer using the basic Frame Buffer commands.

Front interprets the two deflection-encoded 3 degree-of-freedom joysticks to define translations and rotations of the data base. The slider is used to define field-of-view. A simple keyboard interface is used to toggle through other joystick modes, for example, to vary light source orientation. If several database files are specified, the user can translate and rotate one or more relative to the others by selecting the appropriate object using the keyboard interface. The keyboard interface also provides for changing background colors, type of anti-aliasing, and so forth.

Other standard programs are being written by the software team. These include *walkthru*, for walking through architectural models, and *csg*, for editing and viewing constructive solid geometry objects.

4. Algorithms and Performance

In this section, we briefly describe the algorithms which have been implemented on Pixel-planes 4 to date, along with some performance figures.

It is our strong belief that many of these algorithms would not have been implemented on Pixel-planes 4 were it not for the relative ease of programmability of the machine. Programmability is facilitated because: (1) the GP is a uni-processor which performs all the pre-processing computations for Pixel-planes, rather than a pipelined or vectorized unit, and (2) the GP is programmable in a high-level language. We believe that the importance of ease of programmability in such experimental machines cannot be overemphasized.

Additionally, we believe that the importance of Pixel-planes transcends its potential for accelerating graphics applications. It in fact represents a radically new way of looking at graphics algorithms, and as such, we believe it has the potential to change the structure of these algorithms and suggest new *methods* for solving problems.

Fast Polygon Rendering. Pixel-planes was originally conceived as a fast polygon renderer. It can render 35,000 triangles per second, with Z-buffer hidden surface elimination and Gouraud shading. Rendering speeds for polygons with more than 3 edges are somewhat slower; for example, quadrilaterals are rendered about 20% slower.

Shadow Casting. Photograph 2 shows the results of a shadow-casting algorithm developed for the machine by Jeff Hultquist [Fuchs et al., 1985]. Shadows are calculated as a post-processing pass following normal image rendering. In this post-processing pass, the polygons in the scene are processed sequentially, as in the rendering pass. For each polygon, the planes defining the shadow frustum cast by that polygon are broadcast and compared to the Z-buffer value computed during the rendering pass, to define the pixels for which the scene intersects the shadow frustum. For each pixel, a logical union is maintained which defines whether that pixel was shadowed by at least one of the polygons. At the end of the post-processing pass, the intensity of shadowed pixels is attenuated.

Since these shadows are computed using shadow volumes, rather than being treated as separate objects, they can drape naturally over other complex objects in the scene. Scenes with shadows require approximately twice the rendering time of equivalent scenes without shadows. Pixel-planes 4 is the only graphics system, to our knowledge, that can cast true shadow volumes at real-time rates.

Anti-aliasing. We have implemented an algorithm for reducing aliasing effects in polygonal images; some results are shown in Photograph 3. The left-hand image was generated without anti-aliasing -- note the "jaggies" on diagonal edges of polygons. The right-hand image has aliasing effects reduced by taking multiple samples per pixel ('super-sampling'), and then averaging the samples with a filter. After rendering the non-antialiased image, the system continues sampling until it has covered a 7x7-sample filter kernel; however, this super-sampling is interrupted if the user moves an input device. Thus the technique does not diminish the rapid interactive system response; in other words, the anti-aliased version of an image is generated only when the system has time. This is the 'adaptive refinement' concept [Bergman et al., 1986]: the massive computational resources needed for interactive computer graphics should be utilized to improve image quality when the image is static.

Sphere rendering. Fred Brooks observed that since the quadratic part of the equation for a circle is constant for all circles, it can be pre-calculated and stored at each pixel; circles are drawn by describing center and radius in a linear expression -- they are effectively polygons with one edge. The algorithms for hidden-surface elimination and smooth-shading of polygons can be extended to spheres in an analogous way, to generate smooth-shaded inter-penetrating spheres [Fuchs et al., 1985]. The image in Photograph 4 shows the molecule Trimethoprime, an anti-cancer drug under development at Wellcome Research Laboratory, and a complex enzyme it is intended to interact with. This image contains about 1,300 spheres and can be updated 11 times per second.

Image enhancement. John Austin has implemented Stephen Pizer's adaptive histogram equalization (AHE) [Pizer et al., 1984] algorithm on Pixel-planes 4 [Austin and Pizer, 1987]. The left-hand image in Photograph 5 is the original CT scan of a human chest. The data contains a much larger range of intensities than can be displayed or perceived. The right-hand image is contrast-enhanced using AHE. Since this procedure requires a local ranking of each pixel in a 512x512 image, it takes about 2 hours on a VAX11/785 (although an approximation to AHE, 'interpolated' AHE, runs in a few minutes); Pixel-planes 4 executes the algorithm in about 4 seconds.

Constructive solid geometry. Steve Molnar, Greg Turk, and Clare Durand are working on rendering constructive solid geometry (CSG) objects on Pixel-planes 4, modeling them as complex polyhedra, that is, by using planar approximations to convex CSG primitives. Although this does not provide greatly increased rendering speeds over a polygonal boundary representation of the same scene, the method does allow intersections and logical subtractions of CSG primitives (like a hole drilled through a cylinder) to be computed in real-time on the frame buffer. With a boundary representation, offsetting of intersecting objects requires a lengthy computation to compute the new set of polygons.

The algorithm is implemented by broadcasting the planar equations for each front- and back-facing facet of the polyhedron and maintaining 2 separate z-buffers, a minimum buffer and a maximum buffer; these buffers describe the range of z-values which lie inside the primitive at each pixel address [Jansen, 1986]. An example from this work is shown in Photograph 6. Also shown is an image from the host display running a CSG editor, which with which a user can interactively create a CSG object, viewing the rendered object on Pixel-planes and the CSG expression tree on the host monitor.

Mandelbrot and Julia sets. Photograph 7 shows results of an algorithm implemented by Greg Turk for displaying Mandelbrot and Julia sets. It allows the user to explore the 2-D plane of the set interactively; the image is updated at frame rates. Since all pixel processors are enabled virtually all of the time, this algorithm gives far better utilization of the raw computational power of the Pixel-planes system than we have achieved with any other algorithm; to update these images on the 512 x 512 display at 25 Hz, the enhanced memories perform 1300 million 15-bit adds plus 655 million 15-bit multiplies per second.

5. The Next Generation Architecture

We plan to define and implement a new generation of the Pixel-planes architecture. We are currently finalizing the architectural specification for this new system, Pixel-planes 5, and are beginning design of the system.

We expect Pixel-planes 5 to be a dramatically more powerful machine than Pixel-planes 4, for two reasons: first, speed on the algorithms performed by the current machine will be considerably greater, due to the higher clock rates and addition of per-object parallelism; second, the architecture is far less restricted than Pixel-planes 4, with *backing store* and random access to pixels, which should allow whole new classes of algorithms to be implemented. In addition to its enhanced performance, Pixel-planes 5 will be configurable in a variety of ways that allow cost to be traded for performance. The machine will also be physically much smaller than Pixel-planes 4 and will consume far less electrical power.

5.1 Performance Increases for Current Algorithms

Performance on currently implemented algorithms will be increased by using higher clock speeds in the processor enhanced memories and by providing a per-object parallelism in addition to the per-pixel parallelism which forms the basis of the Pixel-planes approach. Additionally, the pre-processing Graphics Processor must have greatly increased capability to support this higher performance.

Higher clock speeds. We plan to run the enhanced memory chips of Pixel-planes 5 at 40 MHz, for a direct 4-fold speed up over Pixel-planes 4, which runs at 10 MHz. This should not be difficult on the enhanced memory chips because of the much smaller silicon geometries. Upgrading the Image Generation Controller (the sequencer which controls the enhanced memory chips) to these higher clock speeds will, however, be a significant challenge.

Per-object parallelism. When rendering polygons, the first thing that Pixel-Planes 4 does is to disable all pixels on the *outside* of the first edge of the polygon. Subsequently, all pixels outside the polygon are disabled, and not until the beginning of the next polygon are any of these pixels' processors performing any useful computation. Since complex databases generally contain mostly very small polygons, the processors of the enhanced memories have a very low utilization. To remedy this, we plan to provide means for processing multiple polygons, lying in disjoint parts of the display screen, simultaneously. This can be thought of as a per-object parallelism superimposed on the per-pixel parallelism of Pixel-Planes 4.

To implement this per-object parallelism, the frame buffer will contain a number of Image Generation Controllers. These IGC's control pixel processors representing disjoint regions of the display screen. Each IGC has a FIFO input buffer. The pre-processor (which computes the A,B,C coefficient sets and frame buffer instructions) broadcasts each primitive only to those IGC's whose portion of the display is covered by that primitive. Since polygons will be quite small, the majority of polygons will be processed by only one IGC. Thus if the IGC's FIFO's are deep enough and if the order of the broadcast primitives is reasonably uncorrelated with their screen position, the multiple IGC's should be able to process several different primitives simultaneously. Simulations with reasonable assumptions about primitive randomization and FIFO depth and typical databases show speed-ups of 4X or better for systems using 16 IGC's.

Pre-processing power. These increases in performance rely upon the availability of a graphics processor capable of computing the required number of coefficient sets for up to perhaps one million polygons per second. Since no single processor can provide this amount of floating-point computational power, some form of parallelism is required. The graphic processor will be implemented using many, perhaps 8 or 16, Weitek XL chip sets of the type used in the Pixel-planes 4 GP, operating in parallel.

5.2 Architectural Enhancements

We plan to enhance the architecture of Pixel-planes 5 in several ways, making it possible to do new classes of algorithms. First, we plan to tightly couple the enhanced memories to a conventional memory *backing store*. Second, we plan to provide a separate conventional frame buffer from which the display screen is refreshed and to which images can be block transferred from the backing store at very high rates. Third, we plan to upgrade the pixel processors of the enhanced memory chips to evaluate quadratic expressions in parallel for all pixels and to have more bits of local pixel memory.

Backing store. Pixel-planes 4's memory organization presents three primary limitations: First, the Frame Buffer's contents are directly scanned out to refresh a screen, but there is no means of reading pixel data back to the host or GP. Second, the only method for addressing an individual pixel (the standard operation of a conventional frame buffer) is to scan convert a 1x1 pixel rectangle, and this can be done no faster than about 3μsec/pixel. Third, there is no communication between pixels, so one pixel cannot directly make use of data stored in a neighbor.

We plan to address these three problems in a unified way, by tightly coupling the processor-enhanced memory chips to dense commercial memories, the *backing store*. We plan to make use of the fast serial port on commercial Video RAM's to support rapid backing store operations to and from our enhanced memory chips, which will have a special I/O port designed to support a serial protocol that complements that of the Video RAM. The primary difficulty in building such a scheme is the unconventional arrangement of our enhanced memories. In frame buffers built from commercial RAMs, the bits of a pixel are spread across multiple memory chips (to increase bandwidth), whereas on our enhanced memory chips, all the bits of a pixel are together on one chip. We plan a simple custom chip to interpret between the different arrangements of the two memories.

The random I/O port of the video-RAM's will be mapped onto the Pixel-planes 5's main data bus, so that pixels can be accessed in the conventional random fashion. Various backing store operations will be supported: pixel data from the enhanced memories can be moved to other parts of the system (to the Graphics Processor, for example) and used in other calculations; data can be written into the backing store in normal random fashion and then transferred into the enhanced memories, perhaps to be processed in vector form; pixel data from the enhanced memories can be shuffled using the random port to simulate inter-pixel communication.

Separate frame buffer. The current Pixel-Planes 4 system implements a 512 x 512 array of physical pixel processors, which directly refresh the display screen, with a one-to-one correspondence between pixel processors and displayed pixels. In Pixel-planes 5, rather than directly refreshing the display from the enhanced memories, we plan to refresh the screen from a frame buffer built from Video RAMs in the standard fashion. Pixel data will be moved in blocks from the enhanced memories to the frame buffer on the system's high-speed bus. This will support higher display resolutions, (1K x 1K and higher) without the necessity of providing greater amounts of the very expensive enhanced memory. Alternatively, lower performance systems can be built with fewer than 512x512 pixel processors.

Improved pixel processors. We plan to enhance the power of each pixel processor in two significant ways: First, the number of bits of local pixel memory at each pixel will be increased to 256 bits, versus the 72 bits per pixel of Pixel-planes 4. Second, the bilinear multiplier tree of Pixel-planes 4 will be replaced by a quadratic unit that evaluates the general 2nd-order polynomial in 2 variables ($Dx^2+Exy+Fy^2+Ax+By+C$) simultaneously for every pixel. This will permit much faster sphere rendering, rendering generalized conic sections, and 'fast' Phong-shading [Bishop, 1986]. This new quadratic tree requires approximately double the silicon area required for the simple bilinear tree of Pixel-planes 4. This, combined with the larger amount of local pixel memory, means that the Pixel-planes 5 enhanced memory chip will retain the 30/70 processor/memory area ratio of Pixel-planes 4.

5.3 Physical Size and Cost Issues

Pixel-planes 4 is physically a very large system, consisting of thirty-two 15" x 15" cards containing 2048 custom VLSI chips for the Frame Buffer, eight 12" x 12" cards for the Image Generation and Video controllers and the GP, and ten 750 watt power supplies. We intend for Pixel-planes 5 to be a considerably smaller, less costly, and less power hungry machine. It will reside in one 19" wide rack and will use no more than 2000 watts of electrical power. These goals will be achieved in several ways.

First, the Pixel-planes approach has the fortunate property that its systems directly benefit, without modification, from advances in semiconductor technology. Recent announcements of 256KBit static memories suggest that enhanced memory chips of at least 64 KBits can readily be built on commercial fabrication lines. We plan to use a 1.2 micron CMOS process, with double metal and polycide, to increase the number of pixel processors on a chip, thereby reducing the number of chips in the enhanced memory array.

Equally important for reducing system size and cost is the use of more aggressive packaging technologies than the dual in-line and pin-grid array packages and through-hole circuit boards used in Pixel-planes 4. The new system will use leadless packages, surface mounted on multi-layered printed circuits, not only to reduce system size but also to help support higher clock speeds.

Finally, we plan to make liberal use of application-specific integrated circuits (ASIC's) in the system to reduce the number of TTL 'glue' chips.

Pixel-planes 5 will be a more modular system than was its predecessor. The pre-processor and enhanced memory array will each be implemented on multiple identical boards; the boards of the system will be connected by a very high bandwidth bus structure. The components can be assembled in a variety of ways to build, for example, small, inexpensive systems with modest performance or large systems with very high performance. The goal is to provide a range of cost/performance with a small set of basic components.

6. Acknowledgements

The work we have reported was funded jointly by the National Science Foundation (grant no. 8300970) and the Defense Advanced Research Projects Agency (contract no. DAAG 29-83-K-0148).

We thank our team of graduate student research assistants, who developed much of the software environment and many of the algorithms described here: Clare Durand, David Ellsworth, Howard Good, Victoria Interrante, Roman Kuchkuda, Steve Molnar, John Rhoades, Brice Tebbs, and Greg Turk.

Our work in system building would have been impossible without our department's Microelectronic Systems Laboratory, its director Vernon Chi, and staff members Mark Monger, John Thomas, and Brad Bennett.

We thank our team mathematician, Jack Goldfeather of Carleton College, for lending mathematical rigor, ideas, curved surfaces, and quadratic expression evaluators.

We thank Mark Kellam and Wayne Dettloff, of the Microelectronics Center of North Carolina, for help with system design and testing and chip fabrication.

We thank our colleague, Professor Frederick P. Brooks, Jr., for years of advice and support and for the sphere rendering algorithm.

We thank our past graduate assistants: Greg Abram, John Cromer, Amarie Helton, Justin Heinecke, Scott Hennes, Cheng-Hong Hsieh, Jeff Hultquist, Alex Melnick, Mary Ranade, and Susan Spach, for years of dedicated creative effort.

We also thank: The MOSIS Project, for IC and circuit board fabrication; John Ousterhout, who has provided tools for the U.S. university VLSI community; Chuck Seitz, whose ideas have greatly influenced our design style [Seitz, 1985]; our colleagues at Xerox PARC, Alan Paeth (now at University of Waterloo), Lynn Conway (now at University of Michigan), and Alan Bell, who collaborated on early designs; Data General, for gifts of cabinets and power supplies; and SCI Inc., for wave soldering services.

6. References

Austin, J. and S. Pizer. 1987. "A Multiprocessor Histogram Equalization Machine," Proceedings of the Xth Information Processing in Medical Imaging International Conference, Utrecht, The Netherlands.

Bishop, T.G. 1986. "Fast Phong Shading," Computer Graphics, **20**(4), (Proceedings of SIGGRAPH '86), pp 103-106.

Fuchs, H. and J. Poulton. 3rd Quarter, 1981. "Pixel-planes: A VLSI-Oriented Design for a Raster Graphics Engine," *VLSI Design*, **2**(3), pp 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, pp 137-146.

Fuchs, H., J. Goldfeather, J.P. Hultquist, S. Spach, J.D. Austin, F.P. Brooks, J.G. Eyles, and J. Poulton. July, 1985. "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-planes," *Computer Graphics*, **19**(3), (Proceedings of SIGGRAPH '85), pp 111-120.

Goldfeather, J., J.P.M. Hultquist, and H. Fuchs. August, 1986. "Fast Constructive Solid Geometry Display in the Pixel-Powers Graphics System," *Computer Graphics*, **20**(4), (Proceedings of SIGGRAPH '86), pp 107-116.

Jansen, F.W. May, 1986. "A Polygon-Halfspace Representation for Rendering Linear Approximations of Curved Surfaces," (submitted for publication).

Kay, A. September, 1977. "Microelectronics and the Personal Computer," *Scientific American*, **237**(3).

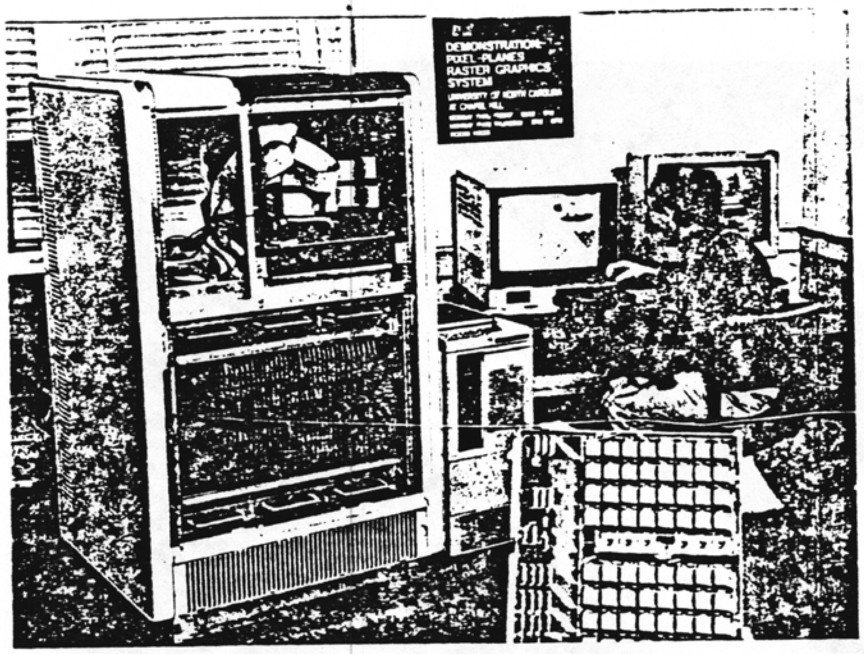
Pizer, S.M., J.B. Zimmerman, and E.V. Staab. 1984. "Adaptive Grey Level Assignment in CT Scan Display," *Journal of Computer Assisted Tomography*, **8**(2), pp 300-305.

Poulton, J., H. Fuchs, J.D. Austin, J.G. Eyles, J. Heinecke, C-H Hsieh, J. Goldfeather, J.P. Hultquist, and S. Spach. 1985. "PIXEL-PLANES: Building a VLSI-Based Graphic System," *Proceedings of the 1985 Chapel Hill Conference on VLSI*, Rockville, MD, Computer Science Press, pp 35-60.

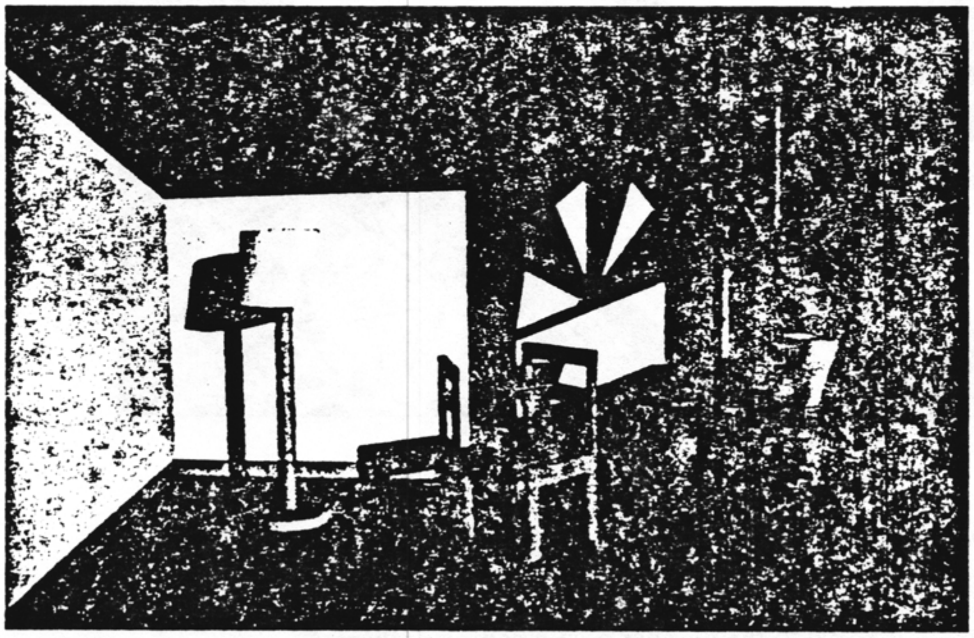
Poulton, J., H. Fuchs, J.D. Austin, J.G. Eyles, and Trey Greer. 1987. "Building a 512x512 Pixel-planes System," *Proceedings of the 1987 Stanford Conference on Advanced Research inVLSI*, Cambridge, MA, MIT Press, pp 57-71.

Seitz, C.L., A.H. Frey, S. Mattisson, S.D. Rabin, D.A. Speck, and J.L.A. van de Snepscheut. 1985. "Hot-Clock nMOS," *Proceedings of the 1985 Chapel Hill Conference on VLSI*, Rockville, MD, Computer Science Press, pp 1-17.

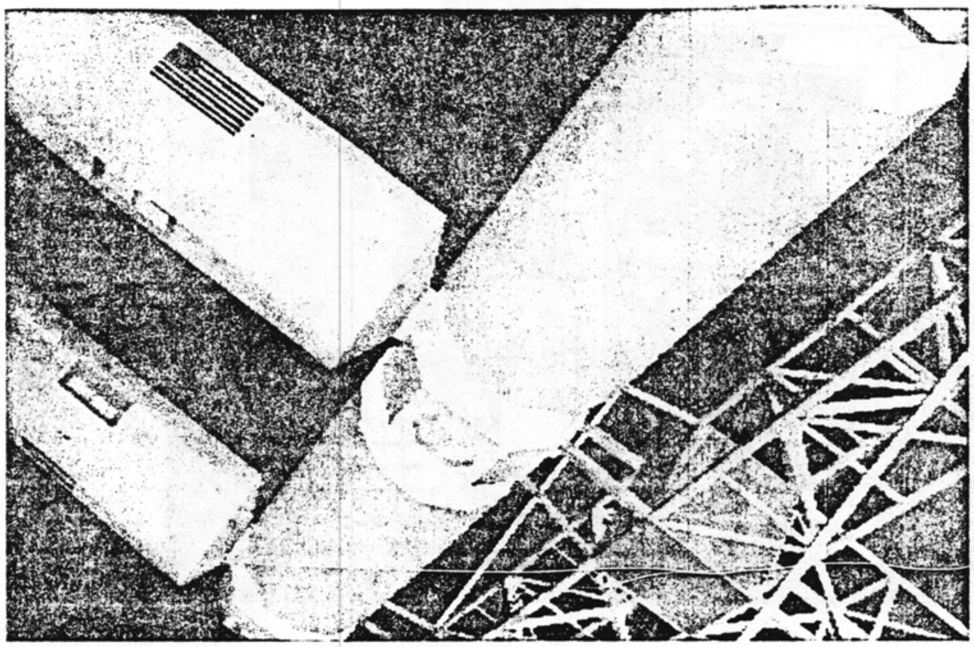
Thacker, C.P., E.M. McCreight, B.W. Lampson, R.F. Sproull, and D.R. Boggs. 1979. "ALTO: A Personal Computer," Xerox Corp.; also in Siewiorek, Daniel P., C. Gordon Bell, and Allen Newell, *Computer Structures: Principles and Examples*, McGraw-Hill, 1982, pp 549-572.

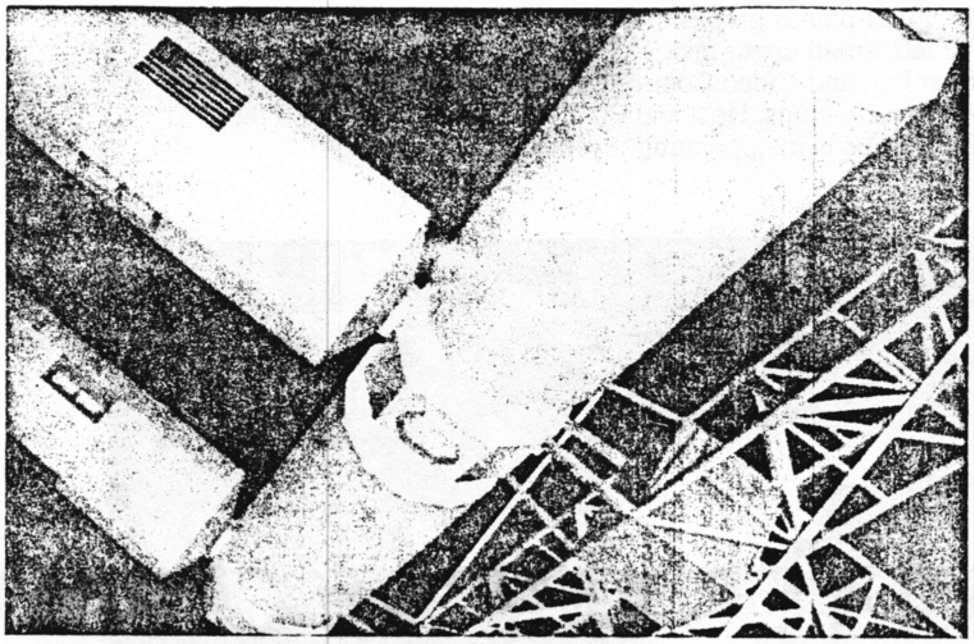


Photograph 1: Pixel-planes graphics system. Cabinet containing custom hardware on the left: small upper rack contains Graphics Processor, Image Generation Controller, and Video Controller; large lower rack contains 2048 logic-enhanced memory chips. Host and Pixel-planes monitor are on the right with PI Henry Fuchs manipulating joysticks.

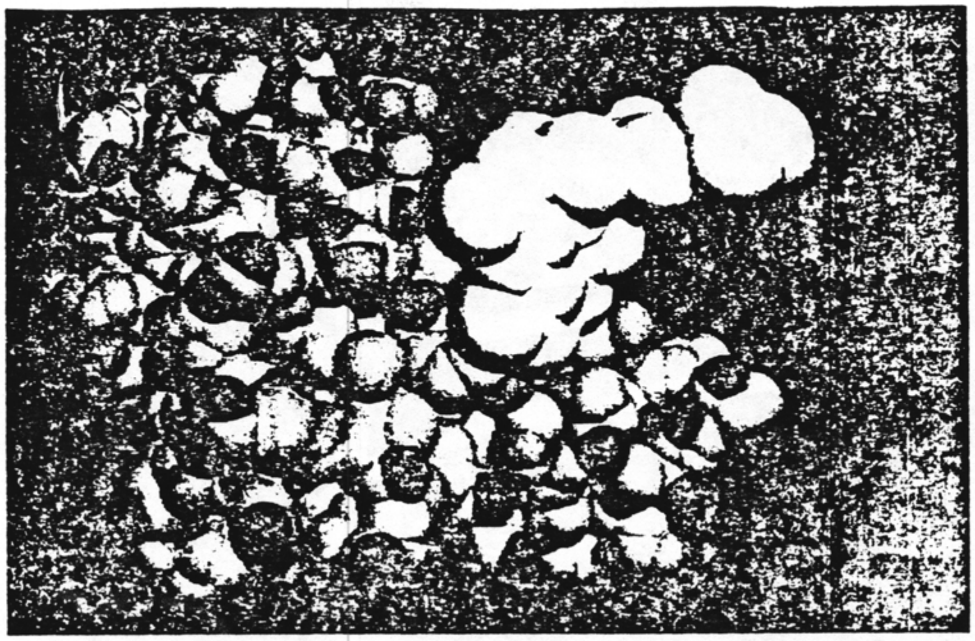


Photograph 2: Polygonal image with true shadow volumes. Updated at frame rates. Database courtesy of Lee Westover.

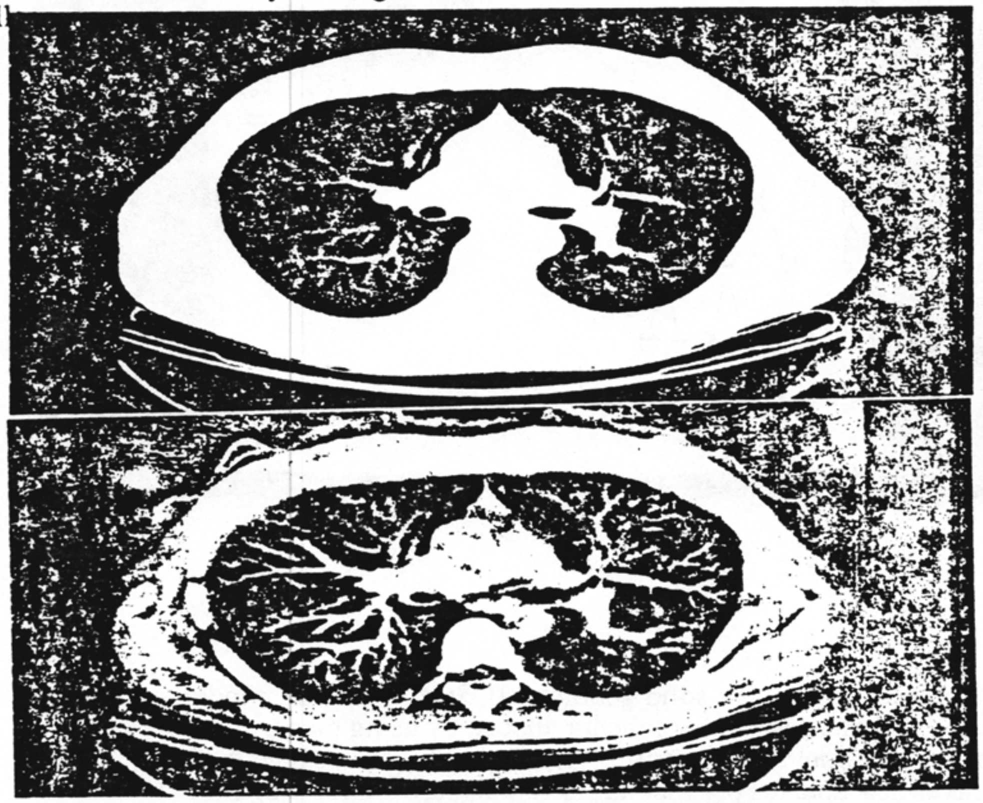




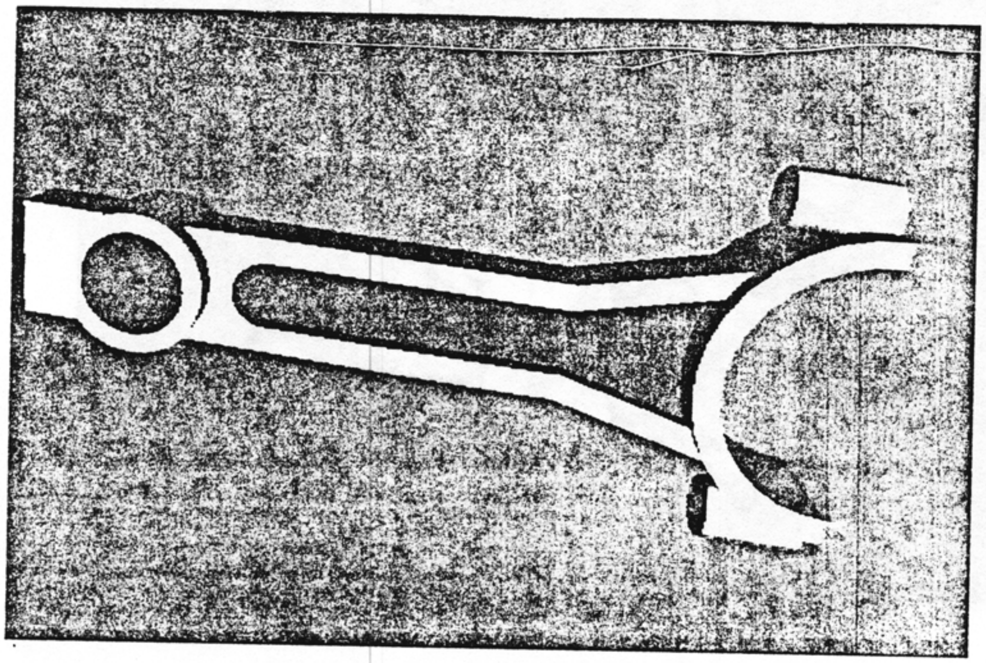
Photograph 3: Polygonal image before (left) and after (right) anti-aliasing. Non-antialiased image updated at 8 frames per second; anti-aliased image generated in about 6 seconds after user releases input devices. Speed penalty is equal to the number of sub-samples taken. A non-antialiased image is produced as the image moves, but when the image becomes static the super-sampling algorithm is applied in successive refinements (with an increasing number of sub-pixel samples). Database courtesy of Don Eyles.

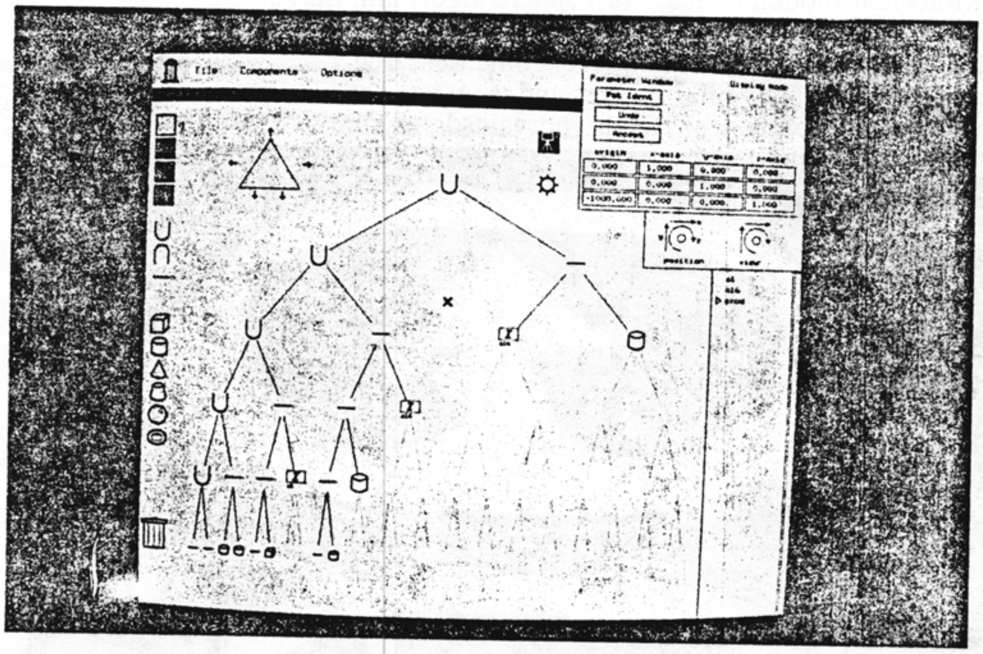


Photograph 4: Molecular modeling using fast sphere algorithm. Large molecule is the protein Dihydro Folate Reductase, smaller molecule (all white) is a drug Trimethoprime under study at Wellcome Research Laboratories. Image contains about 1300 spheres and is updated at 11 frames per second. Database courtesy of Helga Thorvaldsdottir and Burroughs-Well

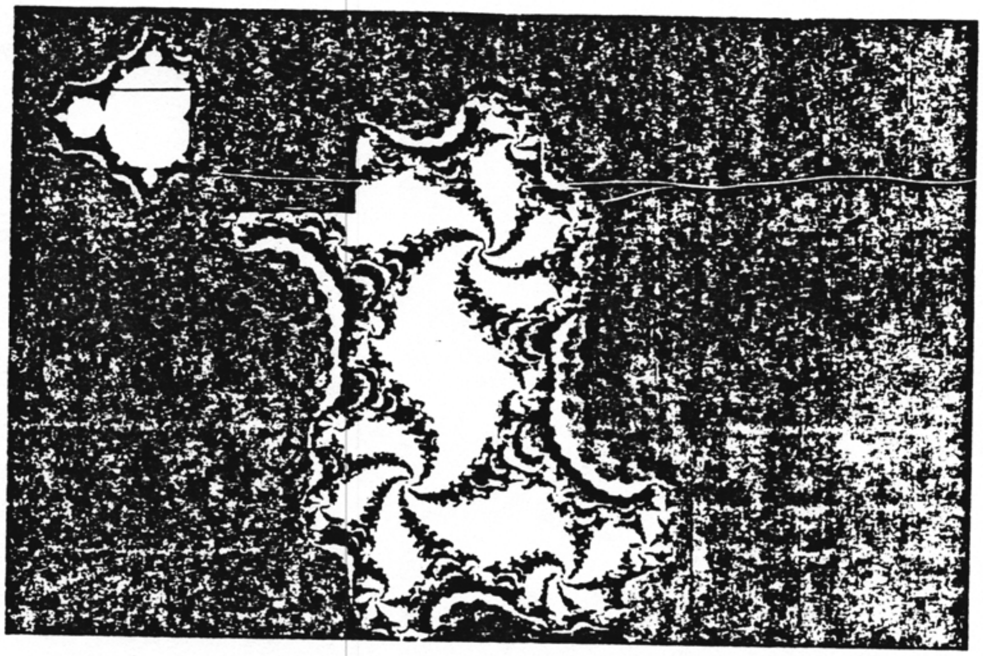


Photograph 5: Original CT image (top) and AHE processed image (bottom). Processed image generated in 4 seconds. Data courtesy of UNC Memorial Hospital Nuclear Medicine.





Photograph 6: Constructive solid geometry (CSG) objects are rendered by modeling them as complex polyhedra, that is, by using planar approximations to convex CSG primitives. Connecting rod generated from 19 primitives at 7 updates per second (left); database courtesy of U.S. Army BRL. Window showing interactive CSG editor on MicroVax II graphics terminal (right).



Photograph 7: Fractal images. Mandelbrot set shown in inset window. Main image shows Julia set derived from point in Mandelbrot set defined by crosshairs in inset. These images are updated at frame rates.