PART TWO. LOW COST IMAGE PROCESSING

Vision is the greatest gift of life. It is also the most complicated interface between an animal and its environment, and with the highest information bandwidth. Image processing is basically a simulation of our vision system, opening a real window in the computer to obtain real information from the real world. Because of the high information bandwidth and the large amount of data to be stored and processes in real time, image processing has been a very expensive proposition. Ten years ago when I first got involved in this subject, an image processing which can handle images of 512 by 512 pixels of 8 bits data cost almost 1 million dollars. Only big government agencies, big companies, and big universities could afford them. The reason was very simple. Image processor depends upon fast RAM memory chips for storage and processing, and needs lots of them, in the order of mega bytes. As the cost of RAM is reduced by 50% every year, the price of image processors are driven down at roughly the same rate. Ten years later, now the cost of a simple image processor is about one thousand dollars, very close to the threshold of pain for personal computer enthusiasts. As the price drops, we will see more people using them to do innovative things unimaginable to computer scientists.

These low cost image processors come in the form of plug-in boards which can be plugged into low cost personal computers and give these computers eyes to see real things. The users thus get powerful tools to manipulate image data, extract meaningful information from the images, and to create new images to convey new ideas and impressions. Many companies are producing image processors for different microcomputers. We have seen products based on DEC's Q-bus, Intel's Multibus, Motorola's VME bus, the old S-100 bus, and more recently, the ubiguitous PC bus.

It is easy to confuse graphic processors with image processors. Most personal computers nowadays has good graphics capability, capable of displaying lines and shapes in grey levels or in color. However, these graphic processors can only display synthetic images generated by the computer, and they can not import or export images very conveniently from or to standard video equipment, like TV monitors, video recorders, and video cameras. To be called an image processor, it must be able to interface to one or more of these video equipment using NSTC TV standard format. A TV frame contains about 500x500 or 250,000 pixels refreshed at a rate of 30 Hz. It translates to 250,000 bytes of data transmitted at a rate of 10 MHz. An image processor must be able to handle that amount of data at that speed, which is not a trivial task. This is the reason why image processors have to be specialized processing elements surrounded by large amount of fast memory.

I had the good fortune to gain access to many of them. The manufacturers generally provide some software with the image processor boards, containing utility to let you use the boards

quickly, and some form of library for those who incline to program them. My tendency is to ignore these software but concentrate on their hardware manuals. Once I learn the internal structures and the interface protocols, I can talk to these boards more fluently in Forth and solve my problem more quickly.

I presented an image processing package for the IP5500 image processor from DeAnza, which is now part of Gould, in the first volume of 'Forth Notebook'. IP5500 was a second generation image processing, a good state of the art system eight years ago with a price tag of about $50,000. I used it to simulated the GAPP parallel processor. Shortly after that it was junked. In this volume I will report my experience with three newer, and much less expensive image processors: CAT 1600 from Digital Graphic Systems, QVG-123 from DataCube, and PIP-1024 from Matrox.

Programming these image processors with Forth is very pleasant. My first step was to find the descriptions on the command register, the status register, and the data register in them. Next, I defined these registers as constants and tried to exercise the command register to do some thing which modified the displayed image on CRT, such as writing a write dot, erasing a line, or putting a character on CRT. These experiments taught me how to converse with the image processor board, so that I could write the basic interface routine. After that, I would write a few demonstration routines to impress people around me. Color look up tables were always fun. Splashing colors on the CRT screen always attracts people's attention. The last fun thing was hooking a TV camera to the image processor and let people process their own portraits. After that, well, get on the project which pays the bill.

VI. DATACUBE QVG-123 IMAGE PROCESSOR

1. QVG-123 IMAGE PROCESSOR BOARD

QVG-123 Board is manufactured by Datacube, Inc. in Peabody, MA. Datacube has specialized in low cost imaging boards for mini- and micro-computers for some time. Lately, it got more ambitious and developed quite sophisticated image processing systems for VME bus computers. QVG-123 was one of its earlier products for DEC Q-bus microcomputers like LSI-11/04 and its family members.

QVG-123 provides a full screen of image memory for computer generated or digitized images. Either black/white or RGB pseudo color display is available, with intensity transformation tables for the video input and the three output channels. A character overlay of 24 lines of 80 characters is also included.

The image memory on the QVG-123 board itself is arranged in the form of 768x512x4 bits. A piggy-back board AF-123 adds 4 bits to each pixel, so that the intensity resolution can be increased to 8 bits or 256 grey levels. The image memory can be accessed by the host computer through a pair or X,Y address registers and a data register. The address registers can be incremented automatically after each access, allowing for high speed sequentially memory access.

One video input source is selected through software from four input channels to the high speed A/D converter for image digitization. The A/D converter runs at a rate of 14.3 MHz with 8 bit of resolution. The input is of the RS-170 format, common to most video cameras and other standard video devices. The timing of the video circuitry can be selected from an internal crystal oscillator, or from an external video sync source. An enhanced phase-lock-loop circuit assures the stability when using external sync source.

The composite video outputs are compatible to EIA standards, either the interlaced RS-170 or non-interlaced RS-420. Separated horizontal and vertical sync signals are TTL compatible. QVG-123 produces three channels of 4 bit video outputs. With AF-123, 8 bit video outputs replace the 4 bit outputs. The character overlay signal is available as an independent output on QVG-123.

2. PROGRAMMING MODEL

The QVG-123 board occupies 9 16 bit registers in the I/O space on the DEC Q-bus. The address is normally set at 176000 to 176020 octal, but can be changed by on board jumpers. AF-123 adds 5 more registers immediately after the QVG-123 registers. The location and bit or field definitions of the QVG-123 registers are shown in Figure 8, and those of AF-123 are shown in Figure 9.

The data register is used in conjunction with the horizontal and vertical position registers to read or write the image memory. Image data can be accessed either by 16 bit words or by 8 bit bytes. Using the data register at 176000, the data transfer is 16 bits at a time. Using the data register at 176001, the transfers are in bytes.

Pan and scroll registers allow the image to be shifted in the horizontal or vertical directions. The character address and character data registers are used to read or write the character overlay memory for displaying ASCII characters on the display.

The command and status register is the most complicated register to deal with. The individual bits in this register controls the memory access mode, horizontal and vertical zoom, selecting video inputs, handling and monitoring interrupts, selecting output look-up tables, and selecting character overlay modes. Most of the programming efforts are concentrated in this register.

QVG-123 also allows individual image memory planes to be protected from accidental writing. Two 8 bit masks in the write enable register can be programmed to enable or disable bit-planes in the image memory to receive or ignore data from the A/D converter or from the host.

The command/status register 2 on AF-123 selects the input channel, the input look-up tables and the output look-up tables. These look-up tables are initialized through the red, green, and blue output LUT registers, and the input LUT register.

To program this QVG-123/AF-123 image processor is to write bit patterns into one or more of these registers, and to monitor the status by reading some other registers. In a Datacube data sheet, there is a outline of several procedures to use this imaging system. The outline is shown in Figure 10.

3. FORTH SOURCE CODE

The host computer used to control this QVG-123/AF-123 board set was a LSI-11/23 clone, manufactured by Scientific Micro Systems in Mountain View, CA. This system was much cheaper than the genuine DEC product. The other advantage in the clones was that they could format floppy disks, while the DEC drives

did not allow this heretical practice, in favor of supporting its users with high quality, and thus grossly overpriced, preformatted disks. In stead of buying the equally overpriced operating system and language tools from DEC, we installed polyForth II on the computer as the programming environment.

The programs shown in Listing 8 were all written under polyForth II. The source code was only recently moved to an IBM PC clone so that shadow screens could be added. However, polyForth II is very close to Forth-79 and Forth-83 standards and the words used in this system were restricted to the small set common to all the recent standards.

Since the shadow screens provide lots of comments on the functions and behavior of the words defined in this system, I will restrict myself only to highlight the structure of the system.

Screen 40 contains the definitions of all the registers in QVG-123 and AF-123. They are constants leaving their register address on the stack for fetching or storing.

Screen 41 tests my ability to write directly into the image memory. It ends with a demonstration program WEDGES which paints a linearly shaded image on the display.

Screen 42 has words which manipulate the look-up tables for pseudo coloring and also for image enhancements. CORRECTION computes an output byte from an input byte, according to an S-shape intensity transformation curve using the contrast and brightness values stored in #GAIN and #OFFSET. RAMP fills a look-up table with values generated by CORRECTION. DARK clears a look-up table to wipe out a channel. The utility words BW load all three channels with the same look-up table so that the eight bit image is displayed in black and white. RED, GREEN, or BLUE writes a ramp function into only one channel and darken the other two channels. The result is a grey scale image with one color, red, green, or blue.

Since the application intended for this system would involve lots of real time graphic display, we needed a very fast routine to draw straight lines. Screens 43 to 47 contain an implementation of Bresenham's line-drawing algorithm, which is fast because it avoids multiplication and division in calculating the cooridnates of points along the straight line between two arbitrary points on the display. Screen 49 has a routine which draws a rectangle from two diagonal points.

Screens 50 to 52 are programs which move data from disk or memory to the image memory and vice versa.

WRAPPING in Screen 53 configures the color look-up tables to display rainbow like color spectrum. The red part or green part of the spectrum can be enhanced by the words MORERED or MOREGREEN.

3DEMO is Screen 54 is an interesting demonstration which draws random boxes on the display with random colors. The word

FILL-IN fills a box with a color by deposit a byte into all the memory locations within a rectangle box.

Screen 55 controls the video A/D converter. BLANKING tests the vertical blanking bit in the command/status register and freezes on frame of image in the image memory before it exits. VIDEO digitizes the real time image and display them at the 30 Hz video rate. Whenever a key on the terminal keyboard is pressed, the displaying loop terminates and the last frame is retained in the image memory.

Screen 56 contains code to generate hardcopy images on a DEC Letterwriter dot-matrix printer. GRAPHICS and TERMINATOR send the control code sequences to Letterwriter to turn the graphic print mode on and off. GCR forces a carriage return in the graphic mode. SEGMENT read 6 bytes at (x,y) position in the image memory and converts them into a byte which will cause the Letterwriter to print 6 dots on paper. GLINE thus prints 6 rows of image, and GBLOCK prints a range of rows. GDUMP dumps the entire image memory to the printer. In converting a pixel to a printable dot, the variable stored in #OFFSET is used as a threshold value. If the pixel intensity is greater than #OFFSET, a dot is printed; otherwise, the dot is not printed.

Figure 11 is my own portrait dumped to the printer. Since the aspect ratio of the printer is not corrected, the image is elongated vertically, making my face much thinner than what it should be.

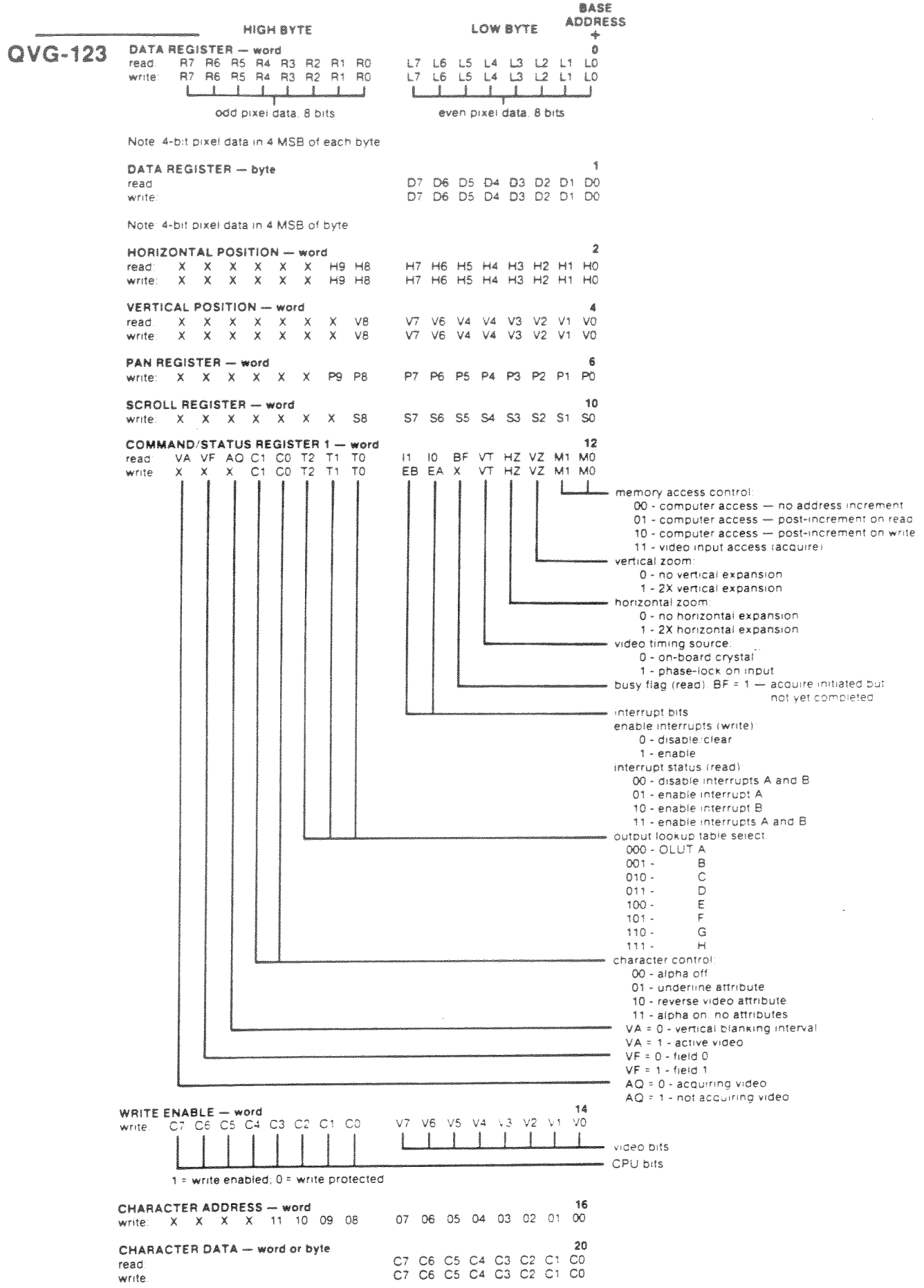


Figure 8. Control and data registers in QVG-123

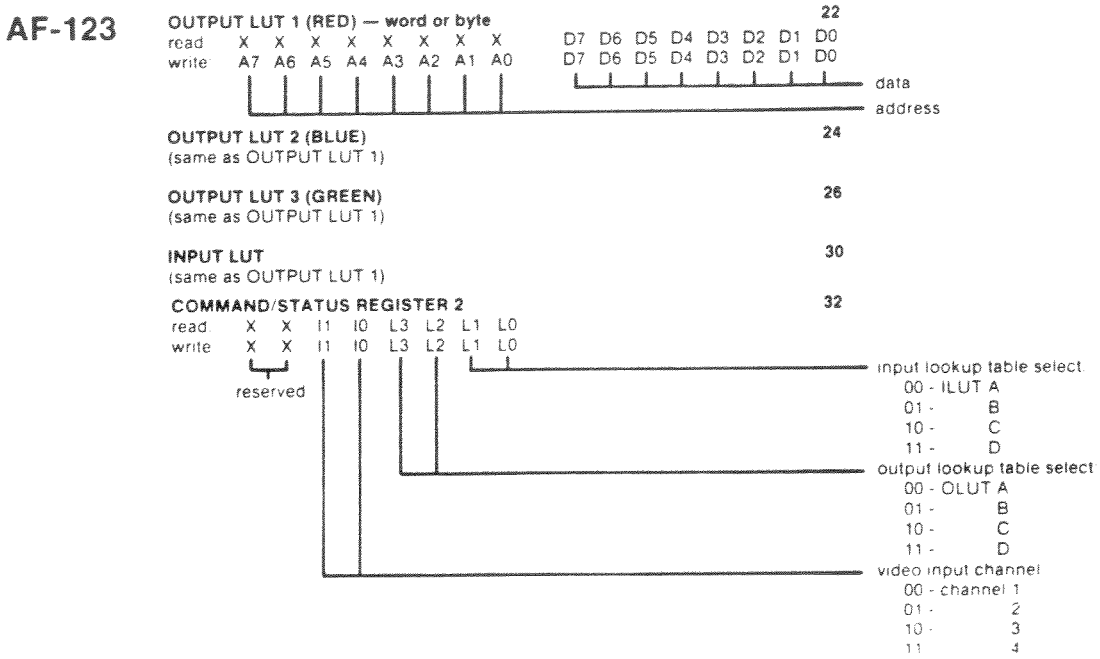


Figure 9. Control and data register in AF-123

PROGRAMMING NOTES

INITIALIZE ROUTINE

Initialize Registers:

1. Write zero to: H Address register;
   V Address register.

Clear screen:

2. Set or clear Command/Status Register 1 bit 4 (VT) to select video timing source;
3. Set Command Register 1 bits 0 and 1 to 10 to select post-increment on write;
4. Write zero to Data Register 393,216 (1400000 OCTAL) consecutive times to clear screen memory.

FRAME ACQUIRE (if video input provided):

1. Set Command/Status Register 2 bits to select an input channel and input and output LUT's;
2. Set Command/Status Register 1 bits 0 and 1 to 11 to enable (video) acquire mode;
3. Set Command/Status Register 1 bits 0 and 1 to 00 to cancel acquire request;
4. Read Command/Status Register 1 Bit 5 for done acquire status. (Bit 5 = 0 when acquire is done.)

REMOTE TRANSMISSION

(Assumes screen memory contains valid image.)

1. Initialize registers, as above;
2. Set Command/Status Register 1 bits 0 and 1 to 01 to select post-increment on read;
3. Read Data Register to obtain pixel data;
4. Write pixel data to desired remote transmission device;
5. Repeat steps 3 and 4 for the desired number of consecutive display memory locations.

REMOTE RECEPTION

1. Initialize registers and clear screen, as above;
2. Set Command/Status Register 1 bits 0 and 1 to 10 to select post-increment on write;
3. Read pixel data from desired remote reception device;
4. Write pixel data byte to Data Register to store in display memory (two 8-bit pixels);
5. Repeat steps 3 and 4 for the desired number of consecutive display memory locations.

Figure 10. QVG-123 programming procedures

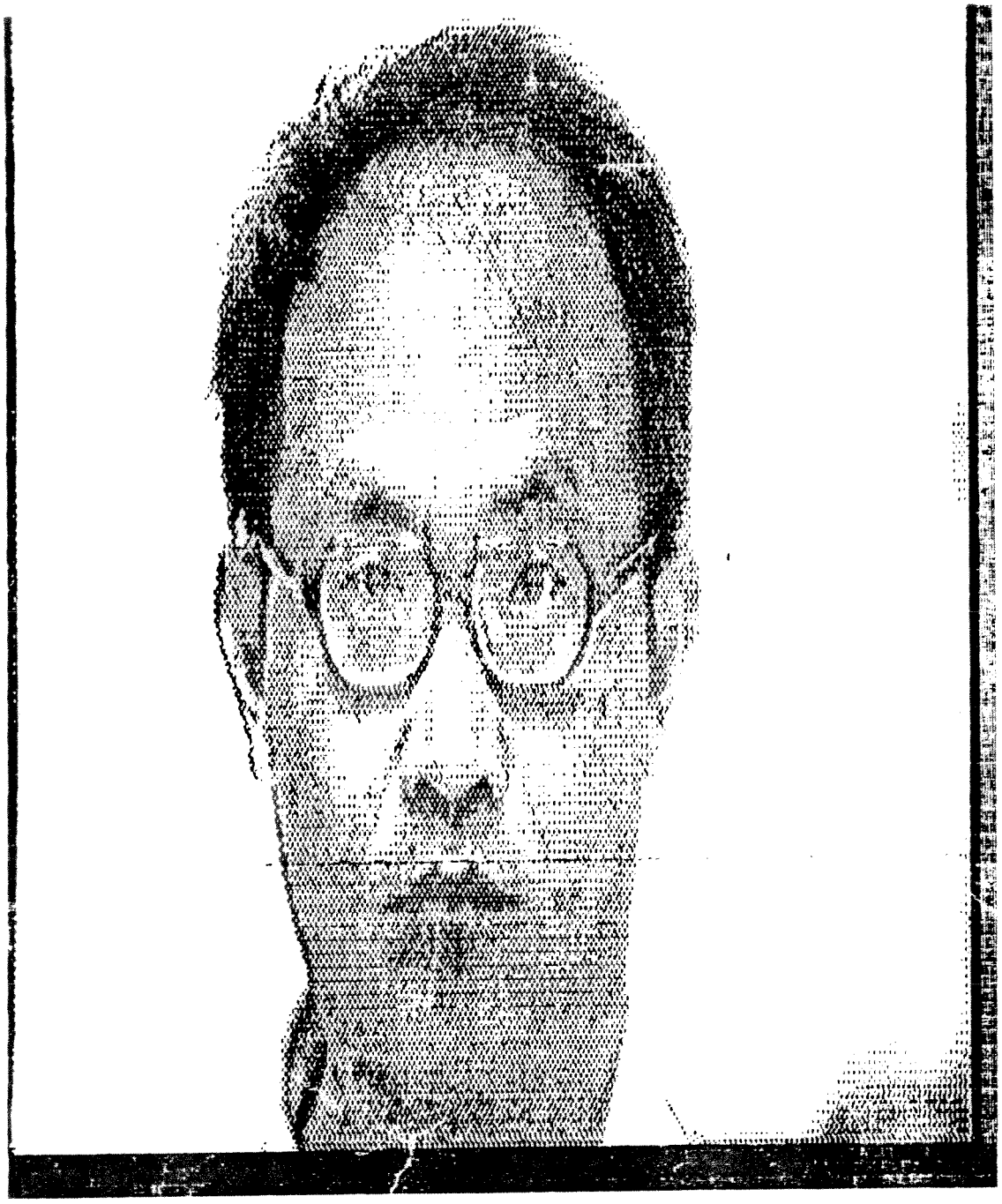


Figure 11. Self portrait

```
 40
 0 ( DATABUS DISPLAY REGISTERS, CHT, 17-MAY-84)
 1 OCTAL
 2 176000 CONSTANT QVGBASE   176000 CONSTANT DWRD
 3 176002 CONSTANT HORZ      176004 CONSTANT VERT
 4 176006 CONSTANT PAN       176010 CONSTANT SCRL
 5 176012 CONSTANT CMND      176014 CONSTANT WENB
 6 176016 CONSTANT CHAR      176020 CONSTANT CDR
 7 176022 CONSTANT ROLW      176024 CONSTANT BOLW
 8 176026 CONSTANT GOLW      176030 CONSTANT ILW
 9 176032 CONSTANT LUTC
10 176001 CONSTANT DBYT
11 DECIMAL
12 -1 WENB !
13
14
15
```

```
 41
 0 ( WRITE TO DISPLAY MEMEORY, CHT, 17-MAY-84)
 1 : WLINE   ( DATA --- )
 2       768 0 DO   DUP DWRD 1+ C!   LOOP   DROP   ;
 3 : WRITE   ( DATA --- )
 4       2 CMND !  ( POST-INCREMENT ON WRITE)
 5       0 HORZ !   0 VERT !
 6       485 0 DO   DUP WLINE   LOOP   DROP   ;
 7 : WEDGE   ( --- )
 8       2 CMND !   0 HORZ !  0 VERT !
 9       485 0 DO   384 0 DO I DWRD !  LOOP   LOOP   ;
10 : PIECE   0 BEGIN DUP DWRD !  1+ DUP 0= UNTIL  ;
11 : WEDGES
12       2 CMND !   0 HORZ !  VERT !
13       3 0 DO   PIECE   LOOP   ;
14
15
```

```
 42
 0 ( OUTPUT LOOKUP TABLES, CHT, 22-MAY-84)
 1 VARIABLE #GAIN                VARIABLE #OFFSET
 2 : GAIN ( N --- ) #GAIN !  ;    : OFFSET ( N --- ) #OFFSET !  ;
 3 : CORRECTION   ( BYTE1 --- BYTE2 , GAIN-OFFSET CORRECTION)
 4       #OFFSET @ -   #GAIN @ *   128 +
 5       DUP 0<   IF DROP 0   ELSE   DUP 255 >
 6       IF DROP 255 THEN   THEN   ;
 7 : RAMP   ( LUT-REG-ADDR --- )
 8       256 0 DO I >< I CORRECTION + OVER !   LOOP   DROP   ;
 9 : DARK   ( LUT-REG-ADDR --- )
10       256 0 DO I >< OVER !   LOOP   DROP   ;
11 : BW     0 LUTC !   ILW RAMP ROLW RAMP GOLW RAMP BOLW RAMP   ;
12 : BLUE   8 LUTC !   ILW RAMP ROLW RAMP GOLW DARK BOLW DARK   ;
13 : RED    16 LUTC !  ILW RAMP ROLW DARK GOLW RAMP BOLW DARK   ;
14 : GREEN  16 LUTC !  ILW RAMP ROLW DARK GOLW DARK BOLW RAMP   ;
15 : GAIN   128 OFFSET   BW
```

```
 140
QVGBASE Base register for QVG-123 board.        23:14 04Oct87cht
DWRD    Data word register.
HORZ    Horizontal position register.
VERT    Vertical position register.
PAN     Pan register.
SCRL    Scroll register.
CMND    Command and status register.
WENB    Write enable register.
CHAR    Character address register.
CDR     Character data register.
ROLW    Red output look up table register.
BOLW    Blue output look up table register.
GOLW    Green output look up table register.
ILW     Input look up table register.
LUTC    Look up table command/status register.
DBYT    Data byte register.
```

```
 141
( WRITE TO DISPLAY MEMEORY, CHT, 17-MAY-84)     23:22 04Oct87cht
WLINE   ( DATA --- )
        Fill one line of image memory with a 16 bit word.
WRITE   ( DATA --- )
        Set post-increment mode and then fill the entire image
        frame with a single word.

WEDGE   ( --- )
        Fill the image memory with a horizontal ramp function.

PIECE   Writing 65536 sequential words to the image memory.
WEDGES
        Fill the image memory with the sequential words.
```

```
 142
( OUTPUT LOOKUP TABLES, CHT, 22-MAY-84)         23:31 04Oct87cht
#GAIN   Gain factor to control contrast.
#OFFSET Offset factor to control brightness.
GAIN    ( N --- ) Set the gain of display.
OFFSET  ( N --- ) Set the brightness of display.
CORRECTION   ( BYTE1 --- BYTE2 )
        Modify the intensity value according to the gain and
        offset setting
RAMP    Write a ramp function to a look up table.
DARK    Write zeros to a look up table.
BW      Write same ramp function to red, blue and green channels
BLUE    Write ramp to blue channel only.  Darken other channels.
RED     Enable red channel only.
GREEN   Enable green channel only.
```

Listing 8. QVG-123 source code

```
 43
 0 ( BRESENHAM'S ALGORITHM, CHT, 18-MAY-84)
 1 ( MINI-MICRO SYSTEMS, NOV. 1983, P. 204)
 2
 3 VARIABLE TESTVAR
 4 VARIABLE A   VARIABLE B
 5 : INIT-DRAW   ( X1 Y1 X2 Y2 --- )
 6       ROT - 2* A !   SWAP - A @ 2DUP SWAP - TESTVAR !
 7       SWAP 2* - B !   ;
 8 : 1DRAW   ( X1 Y1 X2 Y2 --- )
 9        ( X2-X1 > Y2-Y1 > 0 )
10       2OVER 2OVER INIT-DRAW   DROP 1+   ROT DO
11       TESTVAR @ 0<   IF A @   ELSE  1+ B @   THEN
12       TESTVAR +!   I HORZ !   DUP VERT !   -1 DBYT C!
13       LOOP   DROP   ;
14
15
```

```
 44
 0 ( BRESENHAM'S ALGORITHM, CHT, 18-MAY-84)
 1 ( MINI-MICRO SYSTEMS, NOV. 1983, P. 204)
 2 : 2DRAW   ( X1 Y1 X2 Y2 --- )
 3        ( X2-X1 > Y1-Y2 > 0 )
 4       2OVER 2OVER SWAP ROT INIT-DRAW   DROP 1+   ROT DO
 5       TESTVAR @ 0<   IF A @   ELSE  1+ B @   THEN
 6       TESTVAR +!   I HORZ !   DUP VERT !   -1 DBYT C!
 7       LOOP   DROP   ;
 8
 9
10
11
12
13
14
15
```

```
 45
 0 ( BRESENHAM'S ALGORITHM, CHT, 18-MAY-84)
 1 ( MINI-MICRO SYSTEMS, NOV. 1983, P. 204)
 2 : 3DRAW   ( X1 Y1 X2 Y2 --- )
 3       ( Y2-Y1 > X2-X1 > 0 )
 4       2OVER SWAP 2OVER SWAP INIT-DRAW   SWAP DROP 1+   SWAP DO
 5       TESTVAR @ 0<   IF A @   ELSE  1+ B @   THEN
 6       TESTVAR +!   I VERT !   DUP HORZ !   -1 DBYT C!
 7       LOOP   DROP   ;
 8
 9
10
11
12
13
14
15
```

```
 143
( BRESENHAM'S ALGORITHM, CHT, 18-MAY-84)        23:38 04Oct87
( MINI-MICRO SYSTEMS, NOV. 1983, P. 204)

TESTVAR Test value.
A       2(y2-y1)
B       2(x2-x1)
INIT-DRAW   ( X1 Y1 X2 Y2 --- )
       Given two ends, initialize TESTVAR, A and B.

1DRAW   ( X1 Y1 X2 Y2 --- )
       ( X2-X1 > Y2-Y1 > 0 )
       Draw lines with slope less than 45 degrees.
```

```
 144
( BRESENHAM'S ALGORITHM, CHT, 18-MAY-84)        23:39 04Oct87
( MINI-MICRO SYSTEMS, NOV. 1983, P. 204)
(2DRAW)   ( X1 Y1 X2 Y2 --- )
       ( X2-X1 > Y1-Y2 > 0 )
       Draw lines with slope between 135 and 180 degrees.
```

```
 145
( BRESENHAM'S ALGORITHM, CHT, 18-MAY-84)        23:40 04Oct87
( MINI-MICRO SYSTEMS, NOV. 1983, P. 204)
3DRAW   ( X1 Y1 X2 Y2 --- )
       ( Y2-Y1 > X2-X1 > 0 )
       Draw lines between 45 and 90 degrees.
```

Listing 8. QVG-123 source code (cont'd)

```
 46
 0 ( BRESENHAM'S ALGORITHM, CHT, 18-MAY-84)
 1 ( MINI-MICRO SYSTEMS, NOV. 1983, P. 204)
 2 : 4DRAW   ( X1 Y1 X2 Y2 --- )
 3       ( Y2-Y1 > X1-X2 > 0 )
 4       2OVER SWAP 2OVER ROT INIT-DRAW   SWAP DROP 1+   SWAP
 5       DO   TESTVAR @ 0<   IF A @   ELSE  1- B @   THEN
 6       TESTVAR +!   I VERT !   DUP HORZ !   -1 DRVT C!
 7       LOOP   DROP   ;
 8
 9
10 : :CASE CREATE SMUDGE ]    ( USE THE COLON COMPILER)
11       DOES>   SWAP 2* + @   2+ EXECUTE   ;
12
13
14
15
```

```
 146
 ( BRESENHAM S ALGORITHM, CHT, 18-MAY-84)        23:42 04Oct87cht
 ( MINI-MICRO SYSTEMS, NOV. 1983, P. 204)
 : 4DRAW   ( X1 Y1 X2 Y2 --- )
        ( Y2-Y1 > X1-X2 > 0 )
        Draw lines between 90 and 135 degrees.

 :CASE   My favorite case defining word.
        Create positional case statements.
```

```
 47
 0 ( DRAW, CHT, 21-MAY-84)
 1 VARIABLE DECISION
 2 : 4DUP   2OVER 2OVER ;
 3 : ?ABS   ( X1 Y1 X2 Y2 --- , ADD 4 IF /X2-X1/>/Y2-Y1/ )
 4       ROT - ABS >R   - ABS R>   > IF 4 DECISION +! THEN   ;
 5 : ?XY   ( X1 Y1 X2 Y2 --- )
 6       ROT > IF 1 DECISION +! THEN
 7       < IF 2 DECISION +! THEN   ;
 8 : 1SWAPDRAW   2SWAP 1DRAW ;   : 2SWAPDRAW   2SWAP (2DRAW) ;
 9 : 3SWAPDRAW   2SWAP 3DRAW ;   : 4SWAPDRAW   2SWAP 4DRAW ;
10 :CASE (DRAW)   ( X1 Y1 X2 Y2 --- )
11       3SWAPDRAW 4DRAW 4SWAPDRAW 3DRAW 1SWAPDRAW 2SWAPDRAW
12       (2DRAW) 1DRAW   ;
13 : DRAW   ( X1 Y1 X2 Y2 --- )
14    0 DECISION !   4DUP ?ABS 4DUP ?XY   DECISION @ (DRAW) ;
15
```

```
 147
 ( DRAW, CHT, 21-MAY-84)                         23:50 04Oct87cht
 DECISION  A variable determining the direction of a line.
 4DUP    Duplicate top four items on data stack.
 ?ABS    ( X1 Y1 X2 Y2 --- )
        Add 4 to DECISION if /X2-X1/>/Y2-Y1/ .
 ?XY     ( X1 Y1 X2 Y2 --- )
        Set bits in DECISION according to the direction of the
        line given.
 1SWAPDRAW   2SWAPDRAW   3SWAPDRAW   4SWAPDRAW
        Need four other cases to cover 180 to 360 degrees.
 (DRAW)  ( X1 Y1 X2 Y2 --- )
        Big case to draw lines in eight directions.
 DRAW    ( X1 Y1 X2 Y2 --- )
        Real line drawing routine.
```

```
 48
 0 ( CASE, WHITNEY & CONRAD, COMPUTER DESIGN, 21-APR-83, P. 81)
 1 EXIT
 2 : :CASE CREATE SMUDGE ]    ( USE THE COLON COMPILER)
 3       DOES>   SWAP 2* + @   2+ EXECUTE   ;
 4 : LEFT   ." TURN LEFT." ;
 5 : RIGHT   ." TURN RIGHT." ;
 6 : STRAIGHT ." GO AHEAD." ;
 7 :CASE   DIRECTION   LEFT RIGHT STRAIGHT ;
 8
 9
10
11
12
13
14
15
```

```
 148
 ( CASE, WHITNEY & CONRAD, COMPUTER DESIGN, 21-AP23:51 04Oct87cht
 Tests for the case structure word.
```

Listing 8. QVG-123 source code (cont'd)

```
 49
 0 ( LINE DRAWINGS, CHT, 22-MAY-84)
 1 : RECTANGLE   ( X1 Y1 X2 Y2 --- )
 2       4DUP DROP OVER DRAW   2OVER DROP OVER 2OVER DRAW
 3       2DUP >R DROP 2OVER OVER R> DRAW
 4       2SWAP >R DROP OVER R> DRAW    ;
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
 149
( LINE DRAWINGS, CHT, 22-MAY-84)                    23:52 04Oct87CHT
RECTANGLE   ( X1 Y1 X2 Y2 --- )
       Use two diagonal corners to draw a rectangle.
```

```
 50
 0 ( IMAGE DATA MOVEMENTS, CHT, 22-MAY-84)
 1 CODE >6   ( ADDR COUNT ROW COLUMN --- )
 2       HORZ S )+ MOV   VERT S )+ MOV   2 S )+ MOV   0 S )+ MOV
 3       BEGIN   DWRD 0 )+ MOV   2 SOB   NEXT
 4 CODE 6>   ( ADDR COUNT ROW COLUMN --- )
 5       HORZ S )+ MOV   VERT S )+ MOV   2 S )+ MOV   0 S )+ MOV
 6       BEGIN   0 )+ DWRD MOV   2 SOB   NEXT
 7 CODE GFILL   ( DATA COUNT ROW COLUMN --- )
 8       HORZ S )+ MOV   VERT S )+ MOV   2 S )+ MOV   0 S )+ MOV
 9       BEGIN   DWRD 0 MOV   2 SOB   NEXT
10 CODE GERASE   ( COUNT ROW COLUMN --- )
11       HORZ S )+ MOV   VERT S )+ MOV   2 S )+ MOV   0 0 # MOV
12       BEGIN   DWRD 0 MOV   2 SOB   NEXT
13 : WRITE   ( N --- )
14       DUP -1 0 0 GFILL   DUP -1 170 0 GFILL
15       -1 340 0 GFILL   ;
```

```
 150
( IMAGE DATA MOVEMENTS, CHT, 22-MAY-84)             23:57 04Oct87CHT
>6      ( ADDR COUNT ROW COLUMN --- )
       Copy a range of words from host memory to image memory.

6>      ( ADDR COUNT ROW COLUMN --- )
       Copy image memory to host memory.  Move 16 bit words.

GFILL   ( DATA COUNT ROW COLUMN --- )
       Fill image memory with a 16 bit word.

GERASE  ( COUNT ROW COLUMN --- )
       Erase a range of image memory.

WRITE   ( N --- )
       Fill entire image with N.
```

```
 51
 0 ( IMAGE DISPLAY, CHT, 22-MAY-84)
 1 VARIABLE #SCROLL ( CURRENT TOP OF SCREEN )
 2 : SCROLL   ( SCROLL 32 ROWS AND CLEAR THE BOTTOM.)
 3       32 #SCROLL +!   #SCROLL @ SCRL !
 4       32 VERT +!   24576 #SCROLL @ 192 + 0 GERASE   ;
 5 : DISPLAY   ( ADDR ROW COLUMN --- , DISPLAY 1k BLOCK OF DATA.)
 6       SWAP 32 OVER + SWAP DO
 7       2DUP   16 I ROT   >6   SWAP 32 + SWAP   LOOP
 8       DROP DROP   ;
 9 : GLINE   ( BLOCK# --- , DISPLAY 12 BLOCKS OF DATA.)
10       SCROLL 12 0 DO   I OVER + BLOCK
11       #SCROLL @ 192 +    I 32 * DISPLAY   LOOP   ;
12 : DEMO   14 CMND !   0 #SCROLL !   224 VERT !
13       200 0 DO    I GLINE   12 +LOOP   ;
14
15
```

```
 151
( IMAGE DISPLAY, CHT, 22-MAY-84)                    00:10 04Oct87CHT
#SCROLL Top line currently being displayed.
SCROLL  Scroll the image up by 32 rows and blank the bottom
       32 rows.

DISPLAY ( ADDR ROW COLUMN --- )
       Display 1 Kbytes of data in a 32x32 pixel block.


GLINE   ( BLOCK# --- )
       Display 12 blocks of data from disk to image.

DEMO    Display 200 block from the disk to fill the image memor
       Meaningless but pretty demonstration.
       See how source code looks like.
```

Listing 8. QVG-123 source code (cont'd)

```
 52
 0 ( IMAGE DISPLAY, CHT, 22-MAY-84)
 1 : 1DEMO   14 CMND !   0 #SCROLL !   224 VERT !
 2       200 0 DO    I 6LINE   12 +LOOP   ;
 3 : 6BLOCKS    ( ADDR --- )
 4       SCROLL   12 0 DO   DUP   #SCROLL @ 192 +   I 32 *
 5       DISPLAY   1024 +   LOOP   DROP   ;
 6 : 2DEMO   14 CMND !   0 #SCROLL !   224 VERT !
 7       48 0 DO I 1024 * 6BLOCKS 12 +LOOP   ;
 8
 9
10
11
12
13
14
15
```

```
 152
( IMAGE DISPLAY, CHT, 22-MAY-84)                00:10 04Oct87cht
1DEMO   Another silly demostration to verify that the image
        processor and disk work together.
6BLOCKS    ( ADDR --- )
        Display 12 blocks of data from RAM instead of from disk.

2DEMO   Fill image with data from RAM in the host.
        See how object code looks like.
```

```
 53
 0 ( WRAPPING LUT, CHT, 22-MAY-84)
 1 : WRAPPING ( --- )
 2       0 LUTC !   ROLW DARK   6OLW DARK   BOLW DARK
 3       64 0 DO   I >R   I 4 * + 6OLW !
 4       I 64 + >R   DUP 63 I - 4 * +   6OLW !
 5       I 4 * +   BOLW !
 6       I 128 + >R   DUP 63 I - 4 * +   BOLW !
 7       I 4 * +   ROLW !
 8       I 192 + >R   63 I - 4 * +   ROLW !
 9       LOOP   ;
10 : MOREGREEN   ( --- )
11       128 0 DO   I >R   I 2* +   BOLW !
12       128 I + >R   127 I - 2* + BOLW !   LOOP   ;
13 : MORERED
14       128 0 DO   I >R   127 I -  +   6OLW 0 OVER ! !
15       128 I + >R   I  + 6OLW 0 OVER ! !   LOOP   ;
```

```
 153
( WRAPPING LUT, CHT, 22-MAY-84)                 00:14 04Oct87cht
WRAPPING
        Construct a color look up table to display rainbow
        spectrum.

MOREGREEN   Add more green to color look up table.

MORERED     Add more red to the color display.
```

```
 54
 0 ( COLORED RECTANGLES, CHT, 22-MAY-84)
 1 219 LOAD   ( RANDOM NUMBER )
 2 OCTAL
 3 : ARM   OR   1 177560 !   ;
 4 : ?TERMINAL   177560 @ 200 AND   ;       DECIMAL
 5 : FILL-IN   ( COLOR X1 X2 Y1 Y2 --- )
 6       2SWAP 2DUP > IF SWAP THEN   OVER - 2/ SWAP   2SWAP
 7       2DUP < IF SWAP THEN
 8       DO   4DUP   I SWAP   6FILL   DROP
 9       LOOP   2DROP DROP   ;
10 : 3DEMO   ( RANDOM COLORED BOXES )
11       2 CMND !   WRAPPING
12       ARM   BEGIN   256 RANDOM 768 RANDOM 768 RANDOM
13       512 RANDOM 512 RANDOM FILL-IN   ?TERMINAL UNTIL   ;
14
15
```

```
 154
( COLORED RECTANGLES, CHT, 22-MAY-84)           00:18 04Oct87cht
        Need random number generator.

ARM     Enable keyboard interrupt in LSI-11.
?TERMINAL   Return true if any key was pushed.
FILL-IN ( COLOR X1 X2 Y1 Y2 --- )
        Fill a rectangle with COLOR specified.

3DEMO   ( RANDOM COLORED BOXES )
        Continuously draw and color fill rectangles on the
        image display.  Must have this demo because everybody
        else is doing it.
```

Listing 8. QVG-123 source code (cont'd)

```
     55
 6 ( VIDEO INPUT, CHT, 23-MAY-84)
 1 CODE BLANKING ( FREEZE ONE FRAME OF IMAGE.)
 2       BEGIN   CMND TST   0< NOT UNTIL
 3       BEGIN   CMND TST   0<     UNTIL
 4       BEGIN   CMND TST   0< NOT UNTIL
 5       BEGIN   CMND TST   0<     UNTIL
 6       BEGIN   CMND TST   0< NOT UNTIL
 7       CMND 0 * MOV   NEXT
 8 : VIDEO  ( DISPLAY REALTIME IMAGE, FREEZE AT A KEYSTROKE.)
 9       19 CMND !   ARM BEGIN PAUSE ?TERMINAL UNTIL
10       BLANKING   ;
11
12
13
14
15
```

```
  155
( VIDEO INPUT, CHT, 23-MAY-84)                  00:28 04Oct87jrt
BLANKING ( FREEZE ONE FRAME OF IMAGE.)
        MSB bit in the status register is set when QVE is
        not acquiring video image, cleared when it is digitizing
        live video.


VIDEO   ( DISPLAY REALTIME IMAGE, FREEZE AT A KEYSTROKE.
        Enable video digitizer and display real time images
        from the camera until a key is pressed.  Then the last
        frame of image is preserved in memory and displayed.
```

```
     56
 0 ( IMAGE PRINTING TO LETTERWRITER 100, CHT, 25-MAY-84)
 1 HEX
 2 MSB GRAPHICS 1B04 , 315C , 2071 ,
 3 MSB TERMINATOR 1B0C , 5C ,
 4 MSB BOF  2491 , 202E ,
 5 DECIMAL
 6 : SEGMENT   ( ROW COLUMN --- GRAPHIC-CODE )
 7       HERE   0 SWAP   6 OVER + SWAP DO
 8       I VERT !   DENT CE #OFFSET @
 9       ( IF 64 + THEN 2/   LOOP   3F +   ;
10 : GLINE   ( ROW --- , PRINT 6 ROWS OF IMAGE.)
11       BOF   768 0 DO   DUP I SEGMENT EMIT   LOOP   DROP   ;
12 : GBLOCK   ( START-ROW END-ROW --- )
13       SWAP DO   I GLINE   6 +LOOP   ;
14 : GDUMP   PAGE   GRAPHICS   0 500 GBLOCK   TERMINATOR   PAGE ;
15
```

```
  156
( IMAGE PRINTING TO LETTERWRITER 100, CHT, 25-MA00:35 04Oct87jrt
GRAPHICS Turn on Letterwriter's graphic mode.
TERMINATOR Turn off the graphic mode.
BOF     Carriage return in graphic mode.

SEGMENT  ( ROW COLUMN --- GRAPHIC-CODE )
        Grab one byte at the row/column intersection, and
        convert this byte to a printable segment.

GLINE    ( ROW --- )
        Print 6 rows of image on Letterwriter.
GBLOCK   ( START-ROW END-ROW --- )
        Print a range of lines from the image memory.
GDUMP    Print the entire frame on the printer.
```

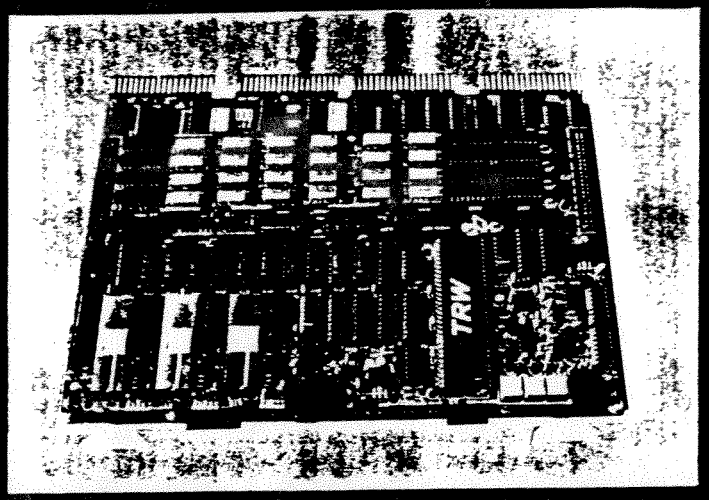
```
     57
 0 ( GRAPHICS TESTING, CHT, 25-MAY-84)
 1 : SEGTST   ( ROW --- )
 2       CR   768 0 DO   DUP I SEGMENT 4 U.R   LOOP   DROP   ;
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
```

```
  157
( GRAPHICS TESTING, CHT, 25-MAY-84)             00:36 04Oct87jrt
SEGTST   ( ROW --- )
        Print 6 rows of image.  Test my understanding of the
        Letterwriter.
```

Listing 8. QVG-123 source code (cont'd)

VIDEO GRAPHICS MODULE MODEL QVG-123 WITH AF-123 EXPANSION

Datacube INC.



Complete video acquire, storage and display generation for the Digital Equipment Corp. LSI-11™ bus. High pixel resolution — 768H x 512V x 4 or 8 bits.

Black-and-white or pseudo-color images, stabilized by an advanced phase-locked loop.

Compatible quad-height modules for PDP™-11/03 and 11/23 processors on the "Q"-bus

Applications — High-resolution digitized video for computer access, and/or computer-driven broadcast-quality graphics display.

- **Video Memory, Display Generation and Output on a Single Module — B&W or R-G-B**
- **AF-123 Expansion Module for Video Input, Expanded Memory and Enhanced Video Outputs**
- **High Pixel Resolution — 768H x 512V x 4 Bits; Expand to 8 Bits with AF-123**
- **Generates Black & White (16 or 256-level illum.) or Pseudo-Color (16 or 256 Color Shades)**
- **DEC™-Compatible Quad Q-Bus Module; AF-123 is Piggyback with no Bus Connector**
- **Full Computer Access to Display Memory — for Analysis, Enhancement, Remote Transmission**
- **Full EIA-Specification Video Input, Output — RS-170 (interlaced) or RS-420 (non-interlaced)**
- **Video Timing Internal (crystal) or Gen-Lock — Software-Selected**
- **Enhanced Phase-Locked Loop for Stability**

INPUT (Conversion) SECTION (Optional AF-123)

- **Real-Time Flash A/D Conversion — 6 or 8 Bits — 14.3 MHz Conversion Rate**
- **Real-Time Computer-Controlled Frame Grab**
- **D.C. Restoration to Correct Input Drift**
- **Four Programmable Lookup Tables for Video Input Translation — Each Channel**
- **Four Video Inputs — Software Selected**

MEMORY SECTION

- **On-Board Full-Screen Digital Memory — 393,216 Pixels x 4 or 8 Bits**
- **Interleaved Memory Access for Video Output and Computer Read/Write**
- **All CSR and Data Registers in I/O Page — Screen Memory Access Through Data Register**
- **X-Y (Indirect) Memory Addressing with Auto-Increment; 8 or 16-Bit Transfers**
- **Write Protection Software Selected for Each Bit Plane Against Computer Access, Video Acquire**

OUTPUT and BUS SECTIONS

- **3 4-Bit Outputs — Grey Scale or R-G-B Display Generation — Real-Time D/A Converters**
- **Eight Selectable ROM Translation Tables for Each Output**
- **Pan, Scroll and Independent Horizontal and Vertical Zoom — 2X factor**
- **Horizontal/Vertical and Composite Sync Outputs**
- **Interlaced or Non-Interlaced Output Format**
- **Character Overlay Generation — 24 Lines of 80 Characters, with Cursor, Reverse Video and Underline Attributes**
- **Status for Vertical Blank and Odd/Even Field**
- **High-Speed Output — 14.3 MHz (4X Color Burst)**

AF-123 (optional):

- **1 or 3 8-Bit D/A Converters — for Grey-Scale or R-G-B Generation**
- **Four Selectable Read-Write Memory Lookup Tables for Each D/A Converter — for Video Translation or Enhancement**

Figure 12. Datacube QVG-123 brochure

DO-LOOP

VII. MATROX PIP-1024 IMAGE PROCESSOR

1. FUNCTIONAL DESCRIPTION

Matrox Electronic Systems, Ltd., is a Canadian company specialized in digital video technology, beginning in the 70's. I remembered that they were the first company producing Video RAM, which was used by microprocessors to generate text display to replace CRT terminals, very expensive at the time.

PIP-1024 is one of Matrox's recent product which is a single board imaging sub-system for IBM PC, XT, and AT computers. It can be though of as a memory buffer containing a digital representation of an image. The image has about 1 million pixels arranged in a 1024x1024 matrix. Any portion of a 512x512 pixels subimage can be displayed on a standard video monitor. Three interfaces exist to the memory buffer: a PC-bus interface allowing CPU or DMA access; an analog input allowing an image to be grabbed from a camera; and an analog output allowing the image to be displayed on a monitor. All three interfaces can be used at the same time without restriction,

PC accesses the buffer using X-Y coordinate addressing. Both random and sequential access are supported, the later using an autoincrement circuitry which increments the X-Y registers after each access. Using this feature, the DMA unit can be used to transfer part or all of the image to or from the PC's memory. An optional interrupt is supplied on completion of the DMA transfer, allowing the CPU to continue with other work while the transfer is taking place.

PIP-1024 digitizes images in real time, 30 frames per second, from the frame grab input port. It accepts RS-170 composite video signals from an external video source, such as a camera, a video recorder, or another PIP-1024. The digitizer runs at 10 Mhz with 8 bit accuracy. A write-protect mask can be used to selectively prevent overwriting of any of the 8 bit memory planes. Protected planes can be used to store graphics, characters, or cursors.

The image is displayed by converting the digital intensity value for each pixel back into analog video signal. The 8 bit pixel intensity is passed through a color look-up table which maps the 256 possible intensity values to 256 colors or shades of grey. The 256 displayable colors are selectable from a 16.7 million color palette. A keying circuit allows the user to overlay the contents of the frame buffer with the incoming signal.

2. THE PROGRAMMING MODEL

PIP-1024 talks to the PC host through a group of I/O ports on the PC-bus. The locations and functions of these ports are summarized in Figure 13. The absolute port addresses can be changed by a jumper on the PIP-1024 board.

Figure 13. PIP-1024 Register Addresses and Functions

Register Offset	Read Register	Write Register
026C	Status	Control 0
066C	Snapshot	Control 1
0A6C	Clear interrupt	Control 2
0E6C		Control mask data
126C	CRTC status	CRTC address
166C	CRTC data	CRTC data
1A6C		Video gain
1E6C		Video set up
226C		Input LUT data
266C		Output blue LUT data
2A6C		Output green LUT data
2E6C		Output red LUT data
326C		X pixel address
366C		Y pixel address
3A6C	Pixel data	Pixel data
02F4(absolute)	Interrupt enable	

It is not appropriate to discuss the bits and fields in all these registers in great details here. One would have to refer to the PIP manual for detailed information. To control the image processor board, the control registers 0 and 1 are used to select functions like enabling the look-up tables (LUT), enable DMA read or write, selecting input channel, internal or external sync, input keying, enabling continuous frame grabbing or snapshot, etc.

The PIP board uses a Synertek SY6845E CRT controller chip to generate all the video timing signals and to control the image buffer memory. This CRTC chip is a very complicated device, having 22 internal registers for its configuration and operation. It is accessible by the PC host through two PC I/O ports at 126C and 166C. This is the same video chip used in IBM PC for color and monochrome displays. Normally the PIP user does not have to touch this chip, as it is automatically configured by PIP board.

PIP is booted into the monochrome mode, in which the look-up tables for the red, green, and blue channels are initialized to a unity function. To display pseudo colors, one will have to rewrite the look-up tables to specify the desired color

transformation. This is done by writing into the proper LUT through the corresponding LUT data register. The byte address in the LUT is selected by writing to the X-pixel register.

To read or write directly from/to the image buffer, the X and Y coordinate of the desired pixel must be written into the X- and Y-pixel address registers, and then the pixel data can be read or written through the pixel data register. Part of the Control 2 register must be used because it takes 20 bits of address to locate a pixel in a 1024x1024 pixel array. Consecutive bytes in the image buffer can be accessed when the Counter-Enable bit in Control 0 register is set. This is convenient to upload an image from PC to PIP or download an image from PIP to PC. Two other bits in Control 0 register control the direction and transferring of data to/from the PC in the DMA mode.

3. PROGRAMMING PIP-1024

Matrox supplies very extensive software for PIP-1024. It includes an interactive control program PIPINT, and several libraries to be linked with FORTRAN or C programs. When PIPINT is loaded into PC, the user can type in many image processing commands to do many things like acquiring images, enhancing the images, upload/down load images, etc. These interactively executable functions are also in the libraries, readily callable from FORTRAN or C programs. Novice users and sophisticated users can all make this board to do useful work in a relatively short time. Matrox did a very good job in supporting the hardware.

In spite of the good software tools supplied by Matrox, I still feel being put in a straight-jacket in using PIPINT. If what you wanted to do is what was programmed in PIPINT, it does a good job. But you always want to do something more sophisticated, or just plainly different. Forth gives you the freedom of tailoring the system to your own needs. The program in Listing 9 is my way of using PIP-1024 through Forth, which is more elegant, I think, than PIPINT, because it can be customized to any application.

When I started to use PIP-1024, I tried to exercise several bits in the control registers which produced visible results on the CRT monitor, to make sure that I had the full control over the system. Several of these functions are shown in Screen 1 of Listing 9. SNAP takes a snap shot and shows an image grabbed from a video camera on the monitor. After I got an image in the image buffer memory, I could then enhance the image by changing the brightness and contrast of the display using the commands GAIN and OFFSET. MASK disabled some bits in the frame grabbing process and produced very interesting effect when two images were superposed in different memory planes.

CTRL0, CTRL1, and CTRL2 are commands to deposit byte patterns into the control registers. They form the basis to build other

more powerful image processing commands. They are used to define the functions in Screen 2.

VREAD and VWRITE in Screen 3 are basic words to copy image data between the PIP image buffer and PC memory. They invoke the command VRAM to set up PIP for autoincrement memory access, so that every time PIP memory is accessed, the X,Y address registers are incremented, making the next byte in image buffer accessible in the next memory cycle.

In one of our applications, we wanted to transfer subimages of 128x128 pixels between PIP and the disk on PC. It required that a subimage anywhere in the image buffer could be accessed. (LINE) in Screen Screen 4 initializes the address registers to access one line of image data at the x,y coordinates. ROWR reads 128 bytes from image buffer to PC and ROWW writes 128 bytes from PC to image buffer. STORE-IMAGE picks a subimage at (x,y) and stores it in the currently opened file. FETCH-IMAGE gets the data from the current file to a subimage in PIP.

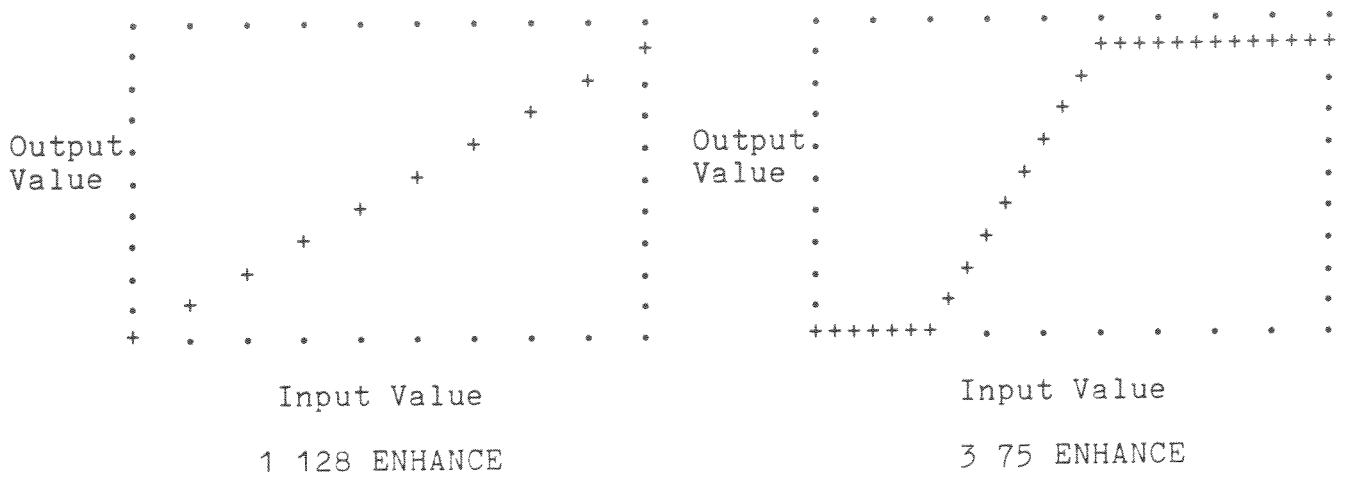
Pseudo color is always fun. Any grey scale image can be made more interesting by coloring different parts with different colors. The commands RED, GREEN, and BLUE write LUT look-up tables to the corresponding channels, assuming that the address of a 256 byte table is on the stack. COLOR-MAP creates a triangle function in the buffer SCRATCH. By fetching different regions in the SCRATCH buffer into the red, green, and blue LUT's, we can create interesting pseudo color schemes. One of such scheme is realized in the command PSEUDO-COLOR in Screen 7, which produces a color sequence of black-red-green-blue-black with increasing pixel values. NORMAL displays a normal black/white image. REVERSE turns black to white and white to black.

ENHANCE is an interesting way to enhance a black/white image. It takes two parameters on the stack: a contrast value and a brightness value. It produces a look-up table with an S-shaped intensity transformation curves: the center of the sloped portion of the curve is specified by brightness value and the slope of this portion by the contrast value. Figure 14 shows the intensity transformation curves produced by the commands

1 128 ENHANCE and 3 75 ENHANCE

The latter command enhances the pixel values between 75-42 and 75+42, the pixels with values from 0 to 42 are displayed in black, and those between 117 and 255 are displayed in white. ENHANCE is very effective in sharpening a poor image in which pixel values are compressed into a narrow band, such as an image taken by a camera with insufficient lighting.

Figure 14. Intensity Transformation Curves.



```
 0
 0 Matrox PIP-1024 utility                                31jul86cht
 1 This package allows you to use PIP-1024 interactively to perform
 2 some elementary image processing functions.  The most important
 3 task in this package is to store and retrieve 128x128 subimages
 4 suitable for GAPP processing.
 5 To load the utility programs, you must first load Forth OS by:
 6
 7 MATROX)f83 do.bl
 8
 9 After Forth is loaded and displays its sign-on message, type:
10
11 1 load
12
13 to load the utility programs.  Then you can type in commands and
14 exercise the PIP-1024 board.
15
```

```
 1
 0 \ Matrox controller                                    31jul86cht
 1 hex
 2 : snap   66c p@ drop ;
 3 : gain   1a0c p! ;
 4 : offset   1e0c p! ;
 5 : mask   0e0c p! ;
 6 : ctrl1   60c p! ;
 7 : ctrl0   20c p! ;
 8 : ctrl2   0a0c p! ;
 9 decimal
10 2 7 thru
11
12
13
14
15
```

```
 2
 0 \ control register 1
 1 variable (quad)
 2 : quad ( n -- )   (quad) !  ;
 3 hex
 4 : cgrab   a5 ctrl1 ;
 5 : freeze  85 ctrl1 ;
 6 : video   81 ctrl1 ;
 7 : keying  8d ctrl1 ;
 8 : quad0   40 ctrl2   0 quad  ;
 9 : quad1   54 ctrl2   1 quad  ;
10 : quad2   68 ctrl2   2 quad  ;
11 : quad3   7c ctrl2   3 quad  ;
12 decimal
13
14
15
```

```
10
                                                          31jul86cht
GAPP needs images in 128X128 format.  These subimages must be
extracted from the 512x512 images normally acquired by PIP board
and displayed on the CRT.  There are four commands defined in
this package which gives you a convenient method to extract
subimages, store them in disk files, and retrieve them.

open (filename.ext)   Open an existing subimage file.
x y fetch-image       Fetch the image from current file and
                      store it in the image memory.  The upper
                      left corner of the image is located at
                      (x,y).  Range of x and y is 0 to 1023.
create-image-file (filename.ext)
                      Create a new subimage file.
x y store-image       Store the subimage whose upper left corner
                      is at (x,y) into the current file.
```

```
11
                                                          31jul86cht
snap      Take a snap shot and store image to current buffer.
gain      Store the gain value into the gain register.
offset    Store offset value into offset register.
mask      Store disabling bit pattern into mask register.
ctrl0     Initialize Control 0 register.
ctrl1     Initialize Control 1 register.
ctrl2     Initialize Control 2 register.
```

```
12
                                                          31jul86cht
(quad)    Variable to store current image buffer number.
quad      Select current image buffer.
cgrab     Continuously grab and display video input.
freeze    Stop cgrab and retain the last image in buffer.
video     Enable video input to be displayed on CRT.
keying    Enable keying for image overlapping.
quad0     Select quadrant 0 as the current buffer.
quad1     Select quadrant 1 as the current buffer.
quad2     Select quadrant 2 as the current buffer.
quad3     Select quadrant 3 as the current buffer.
```

Listing 9. Matrox PIP-1024 utility

```
      3
 0 \ memory read and write
 1 hex
 2 : vram ( addr -- )
 3    100 /mod   3b8c p!   32bc p!   40 ctrl0 ;
 4 : vread ( vadd buf count -- )
 5    rot vram   0 do 3a8c p@ over c!   1+ loop   drop ;
 6 : vwrite ( vadd buf count -- )
 7    rot vram   0 do dup c@ 3a8c p!   1+ loop   drop ;
 8 decimal
 9
10
11
12
13
14
15
```

30jul86cht

```
      13
                                                    31jul86cht
vram      Setup the VRAM registers for autoincrement memory
          accessing.
vread     Read a rangle of image data from vadd and store them
          in a buffer starting at buf.  Count bytes will be
          read.
vwrite    Write a range of data from memory buffer starting at
          buf to image memory starting at vadd.  Count bytes
          will be written.
```

```
      4
 0 \ 128 by 128 windows
 1 hex
 2 : (line) ( x y -- )   40 ctrl0
 3    >r   100 /mod swap 32bc p!   2 /mod swap
 4    r> 80 /mod >r   2* + 3b8c p!
 5    r> swap if 8 + then   (quad) @ 10 * 42 + + ctrl2   ;
 6 : rowr ( adr count x y -- )
 7    (line)   0 do   3a8c p@ over c!   1+ loop   drop ;
 8 : roww ( addr count x y -- )
 9    (line)   0 do   dup c@ 3a8c p!   1+ loop   drop ;
10 decimal
11
12
13
14
15
```

31jul86cht

```
      14
                                                    31jul86cht
(line)    Setup VRAM, CTRL0 and CTRL2 registers for auto-
          incrementing image memory accessing.  (x,y) on the
          stack is the starting coordinates in the 1024x1024
          image memory.  Thus we can address any pixel in the
          PIP memory.
rowr      Read one row of image data and store them in a buffer.
          x and y are the starting coordinates.  addr and count
          define the location and length of the memory buffer.
roww      Write one row of data from the memory buffer to the
          image memory.  Arguments are the same as for rowr.
```

```
      5
 0 \ 128x128 image files
 1 : create-image-file   16 create-file ;
 2 : store-image ( x y -- )
 3    128 0 do   i 8 /mod block   swap 128 * +   128
 4       2over rowr   update   1+
 5    loop   2drop   flush ;
 6 : fetch-image   ( x y -- )
 7    128 0 do   i 8 /mod block   swap 128 * +   128
 8       2over roww   1+   loop   2drop ;
 9
10
11
12
13
14
15
```

31jul86cht

```
      15
                                                    31jul86cht
create-image-file
          Create a 16 Kbyte file to store a 128x128 subimage.
store-image
          Store a 128x128 subimage into the currently openned
          subimage file.  The coordinates (x,y) on stack defines
          the upper-left corner of the subimage.  Range of x
          and y is from 0 to 1023.
fetch-image
          Retrieve a subimage from the current file and store
          it in the image memory.  The upper left corner of the
          subimage is determined by x and y on the data stack.
Examples:
create-image-file test1.pix   300 400 store-image
open test1.pix    100 100 fetch-image
```

Listing 9. Matrox PIP-1024 utility (cont'd)

```
      6
 0 \ Pseudo Coloring                                    31jul86cht
 1 hex
 2 : lut#  ( n -- )   40 * ctrl0   ;
 3 d000 constant scratch
 4 : ramp   100 0 do  i dup scratch + c!   loop ;
 5 : color-map  scratch 200 erase
 6    40 0 do  i 2* 2* dup   scratch i + 80 + c!
 7       scratch 100 + i - c!   loop   ;
 8 : red ( addr -- )   0 326c p!
 9    100 0 do   dup c@ 2e6c p!  1+ loop   drop ;
10 : green ( addr -- )   0 326c p!
11    100 0 do   dup c@ 2a6c p!  1+ loop   drop ;
12 : blue ( addr -- )   0 326c p!
13    100 0 do   dup c@ 266c p!  1+ loop   drop ;
14 decimal
15
```

```
      7
 0 \ Pseudo Coloring                                    31jul86cht
 1 hex
 2 : normal   ramp   scratch dup red   dup green   blue ;
 3 : reverse   scratch 100 0 do   ff i - over c!   1+ loop drop
 4    scratch dup red   dup green   blue ;
 5 : pseudo-color   color-map   scratch dup blue
 6    40 + dup green   40 + red  ;
 7 : enhance ( contrast brightness -- )
 8    100 0 do   2dup   i swap - *   80 +   ff min 0 max
 9       scratch i + c!   loop   2drop
10    scratch dup red   dup green   blue ;
11 decimal
12
13
14
15
```

```
      8
 0 \                                                    31jul86cht
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
     16
 0 \ Pseudo Coloring                                    31jul86cht
 1 hex
 2 : lut#  ( n -- )   40 * ctrl0   ;
 3 d000 constant scratch
 4 : ramp   100 0 do  i dup scratch + c!   loop ;
 5 : color-map  scratch 200 erase
 6    40 0 do  i 2* 2* dup   scratch i + 80 + c!
 7       scratch 100 + i - c!   loop   ;
 8 : red ( addr -- )   0 326c p!
 9    100 0 do   dup c@ 2e6c p!  1+ loop   drop ;
10 : green ( addr -- )   0 326c p!
11    100 0 do   dup c@ 2a6c p!  1+ loop   drop ;
12 : blue ( addr -- )   0 326c p!
13    100 0 do   dup c@ 266c p!  1+ loop   drop ;
14 decimal
15
```

```
     17
 0 \ Pseudo Coloring                                    31jul86cht
 1 hex
 2 : normal   ramp   scratch dup red   dup green   blue ;
 3 : reverse   scratch 100 0 do   ff i - over c!   1+ loop drop
 4    scratch dup red   dup green   blue ;
 5 : pseudo-color   color-map   scratch dup blue
 6    40 + dup green   40 + red  ;
 7 : enhance ( contrast brightness -- )
 8    100 0 do   2dup   i swap - *   80 +   ff min 0 max
 9       scratch i + c!   loop   2drop
10    scratch dup red   dup green   blue ;
11 decimal
12
13
14
15
```

```
     18
 0 \                                                    31jul86cht
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
31jul86cht                                                    31jul86cht

lut#        Select one among 8 sets of look-up-tables.
scratch     A scratch buffer area used to store lut table before
            it is loaded into PIP board.
ramp        Generate a ramp function in scratch buffer to load
            normal or unmodified LUT's.
color-map   Generate a special function in scratch buffer to
            build rainbow like LUT's for pseudo coloring.
red         Build red LUT with specified data buffer.
green       Build green LUT with specified data buffer.
blue        Build blue LUT with specified data buffer.
```

```
31jul86cht                                                    31jul86cht

normal      Build a normal set of LUT's in the current LUT map
            to display data as stored in the image memory.
reverse     Display the image data reversed; i.e., high image val
pseudo-color   Display the image data using pseudo coloring.
            The color scheme is black-red-green-blue-black as
            the image data value increases from 0 to 255.
enhance     Enhance the image according to the contrast and
            brightness values specified on the stack.  For a
            normal display, the command is:
            1 128 enhance
            To enhance an image whose pixel value center around 75
            with twice the contrast, type:
            2 75 enhance
```

Listing 9. Matrox PIP-1024 utility (cont'd)

matrox
electronic systems ltd.
1055 ST. REGIS BLVD., DORVAL, QUE., H9P 2T4, CANADA
TEL.: (514) 685-2630 TELEX: 05-822798

PIP-512

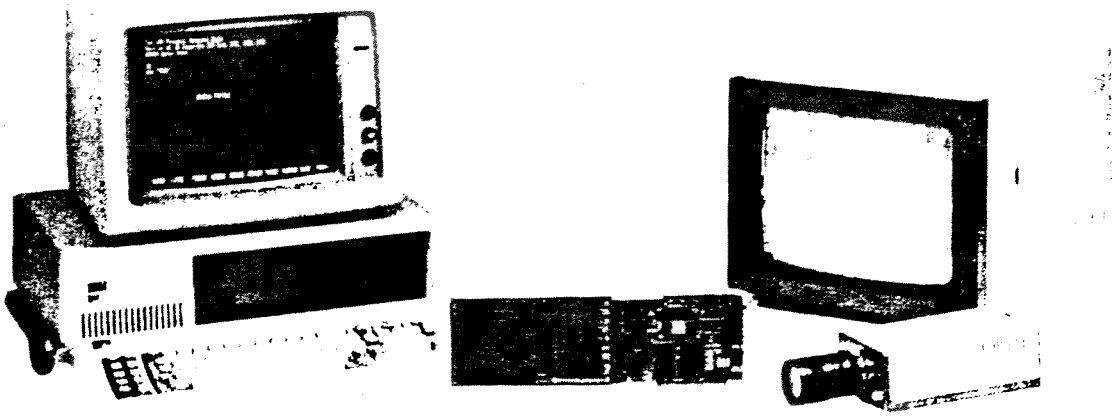
REAL-TIME IMAGE DIGITIZER FOR THE IBM PC

- 512 × 512 display resolution
- 8 bit/pixel
- Optional 1024 × 1024 image buffer
- 16.7 million color lookup-table
- Video keyer & Write mask
- Continuous or one-shot frame grab
- IBM PC, AT and XT compatible
- CMOS design requires only 15 watts
- Occupies a single expansion slot
- 8-bit flash frame grabber
- Internal or gen-lock sync
- Low cost

The MATROX PIP-512 is a full-feature image digitization and display module for your IBM PC, AT, XT or plug-compatible computer. This card has all the high-performance features characteristic of top-of-the-line image processing systems such as real-time 8-bit digitization, a 512 × 512 video buffer, multiple input and output lookup tables, a write-protect mask, and color display.

Equally important, the PIP-512 integrates easily into a PC system. It occupies only one expansion slot and consumes a mere 15 Watts of power. Diagnostic support is built into the hardware. As a result, the diagnostic program, PIP-TEST, is capable of detecting over 95% of all possible problems.

PIP-EZ, a software package conceived to help the programmer to develop an application, is supplied free of charge. PIP-EZ consists of an installable device driver under PC-DOS and applications libraries for all major DOS language (BASIC, PASCAL, FORTRAN, and C).



Your Exclusive Rep/Distributor

Figure 15. Matrox PIP-512 brochure

• PIP-512 FEATURES •

High Speed A/D Converter	A flash A/D converter digitizes RS-170/330 signals in real-time (1/30 sec) to 256 discrete intensity levels.
Sync Control	Both an internal crystal generated sync and PLL genlock to an external signal are supported.
Display Resolution	512 × 512 pixels
Bits/pixel	8
Refresh Rate	50Hz or 60Hz interlaced
Image Buffer Capacity	PIP-512: 512 × 512 PIP-1024: one 1024 × 1024 frame or four 512 × 512 frames
Input Lookup Tables	Eight independently-selectable lookup table maps.
Color Output	Three lookup tables and three D/A converters allow for the display of 256 colors or shades of grey from a palette of 16.7 million colors.
Transparent Memory Access	The Video RAM can be accessed at all times without causing interference on the screen.
Write Protect Mask	Individual planes can be protected from overwriting. Protected planes can be used for graphics, and text.
Pan & Scroll	The PIP-512 provides programmable roam capabilities to a resolution of 8 pixels horizontally and 16 lines vertically.
DMA Capability	Any DMA channel can be used to copy data to and from main memory. An interrupt on completion allows the CPU to continue with other work during the transfer.
I/O Interface	Sixteen I/O registers are used to control each PIP-512. Only one location is taken up in the 10-bit I/O map used by older controllers to ease integration.
Power Consumption	Extensive use of CMOS technology keeps the power consumption down to a mere 15 Watts.
Applications Software	PIP-EZ, a driver package compatible with PC-DOS (versions 2.0 and above) and all major languages (BASIC, C, PASCAL and FORTRAN) aids the user to bring up an application quickly and painlessly.
Diagnostics	A sophisticated diagnostic program aided by hardware support detects more than 95% of all possible faults without the use of either a camera or a monitor.

Figure 15. Matrox PIP-512 brochure (cont'd)

VIII. CAT 1600 IMAGING SYSTEM

1. INTRODUCTION

It seemed to be a long time ago that I joint this project which had required a imaging and displaying system to show wave front images from a optical system. The image was rather simple, consisting of 1024 bytes arranged in an array of 32 by 32 pixels. The data were color coded and displayed on a color CRT monitor in a checkerboard form. At that time, MassComp in Massachusetts had rolled out their MC500 series workstations, and one of them was designated to be the displaying device for this experiment. This MassComp computer was run by three 68000 CPU's, and a few other proprietary processors and was considered the most sophisticated laboratory computer money could buy. Its operating system was an undiluted Unix, supporting both C and FORTRAN. It also had a STD based front end to interface to real world through A/D, D/A, parallel and serial cards. A quite extensive package of software was also developed to perform data acquisition, displaying and analysis tasks.

With such horse power packed in a nice looking box, topped by a high resolution color monitor, mouse, and keyboard, it was thought that displaying such a simple image was a easy piece of cake. However, after months of work, enlisting the best C programmers we could find, the best results obtained was that the display could be refreshed at the rate of about 1.3 Hz and no more. The reason was that this computer was designed principally for graphics, not for imaging. After the 1024 byte data block was received, one of the 68000 had to be dedicated to color fill the checkerboard, and their were a lots of pixels to fill.

About that time, a colleague showed me a fattened microcomputer he used for some optics experiments. It was again a 68000 based computer, built by Ergonomic Research Group in Oregon. It was again a Unix machine. However, the interesting thing about this computer was that when it was cold booted, it displayed the strange message "ok". There ought to be something fishy there. Some more probing convinced me that there was a genuine Forth underneath, whose sole function was to bring the Unix in from the hard disk. A few pages of scattered documentation showed that the Forth was derived from figForth model with significant extensions, like 32 bit stacks and 32 bit addressing capability. The CPU and 1 Megabytes of memory were housed in a S100 card cage, with an extra card set, CAT 1600 from Digital Graphic Systems, Inc. in Palo Alto, CA. The CAT 1600 was a full color, real time imaging system which could digitize and display images from a video camera.

Promising that I could improve the speed of data displaying, I was given ownership of this computer to prove it. Prove it I did. I threw away the Unix and studied the strange Forth came in a pair of PROM's without much documentation. In about a month, we had enough code developed to hook it to the main experiment. With CAT 1600, the data refreshing rate was raised to 40 Hz, which was too fast for the host minicomputer to keep up with it. When people tried to increase the transfer rate at the host side, the host computer crashed at about 20 Hz. So we ran at about 15 Hz and everybody was very happy. The CAT 1600 could be pushed to about 100 Hz, but nobody cared and even if I did, nobody would notice any difference. So I stopped.

I never bothered to tell people why CAT 1600 outperformed the much more sophisticated, and more expensive, MassComp workstation. It was kind of cheating. As a system designed specifically for imaging purposes, CAT 1600 was capable of zooming at a wide range of zoom ratios. I expanded each pixel by a factor of about 8, making the checkerboard large enough to be seen at a distance. At this zoom ratio, I only had to write each square once, unlike MassComp which had to fill every pixel in the checkerboard.

Unfortunately, I learnt lately that this system became sick. Apparently the 68000 CPU quit. The worst thing was that ERG was no longer in business, and nobody knew how to fix it. It was eventually junked. People in that project were trying use the MassComp again. Losing a computer is like losing a child, after you put so much of yourself into it. It is very sad. I feel like writing an obituary now.

Here I will show you some of the Forth code used to control the CAT 1600 imaging system. I cannot find the original listing. The copy, though still legible, is not very clear. My apologies. I cannot even write shadow screens with the source screens to make them more understandable, but I will try to explain what is not obvious.

2. CAT 1600 IMAGING SYSTEM

Digital Graphic Systems pioneered low cost imaging systems as early as 1977. CAT 1600 was developed as board sets to plug into the motherboard of a standard S100 host computer. It contains a large image memory and all the circuitry necessary to digitize B/W or color video signals in real time, to store and process the digital image data, to make the data accessible to the host processor, and to display the digital image on a video screen in gray levels or in full resolution RGB color. A dedicated 8086 image processor is supported on board with up to 48K of fast static RAM and 64K or PROM.

The dual port image memory provides for up to three 512x512x8 bit images or one 512x512x24 bit real color image. The image memory can be accessed at any time, either by the display processor or by the host processor, for software image generation or

analysis of the digitized images.

Other important features include: NSTC standard video input and output, genlock on external video or sync, flexible color look up tables, pixel by pixel scroll and pan, smooth zoom from 1:1 to 1:31.68, etc., etc.

The most interesting feature is the image processing firmware coming in on-board PROM's. About one hundred firmware routines in this library can be called by the host processor using simple commands. This library relieves the host processor from lower level imaging tasks, which significantly speeds development work and increases the overall system performance.

Communication between the host and CAT 1600 is done via two channels: the host I/O port address space and the host memory address space. In the I/O space three consecutive word addresses are used. They are the Command Register, the Data Register, and the Status/Reset Register. The command register is used by the host to select one of the firmware functions. If the firmware function needs parameters, they are passed through the data register. If a function returns results, they are read also by the host through the data register. The host monitors the activity in CAT 1600 by reading the status register. When the status register is written by the host, CAT 1600 will reset.

In the host memory space, an addressing segment, called a data 'window' is assigned by CAT 1600 for direct data exchange. Although it is possible to move all image data through the I/O registers, throughput and speed are greatly improved if the data is moved through the data window.

3. FORTH SOURCE CODE

Controlling CAT 1600 with Forth is rather straightforward, like controlling any other electronic equipment. One has to first establish communication with CAT in monitoring its status register, sending commands to the command register, and passing data to and from the data register. Since the image memory is mapped in the addressing space of the 68000 system, one should also establish the proper protocol to access the image memory directly.

Listing 10 has the CAT 1600 source code. Screen 101 has the elementary code which allows the 68000 to control the CAT 1600.

The three I/O registers are mapped to the memory locations hex FFFFC0. , FFFFC2. , and FFFFC4. The registers are named appropriately DATA for data register, CMD for command register, and STAT for status register. They are all defined as 32 bit constants to return addresses in the 68000 addressing space.

READY is used to wait until CAT finishes its last task and resets the busy bit in the status register. It will abort if made waiting too long. INITCAT resets CAT 1600 by writing to the status register. INWRD reads a word from the data register. OUTWRD writes a data word to the data register, and OUTCMD writes a command to the command register. STATUS returns the contents in the status register. All other Forth commands to CAT 1600 are constructed from these simple interface words.

COLOR was my first attempt to converse with CAT 1600. Command 10 is to fill the entire screen with a byte to produce a flat colored screen. OA CMD sends the command followed by OUTWRD to specify the color. DEMO2 puts COLOR in a loop to test all the colors available.

Screen 102 shows code to write directly into the image memory. The image memory is mapped to the 68000 address space starting at hex 600000., which is defined as IMAGE. To access image memory directly, one has to first enable DIA (Direct Image memory Access) by giving command 74. After the memory accessing is completed, it is necessary to disable DIA (command 0) before issuing other image processing command; otherwise, CAT would abort.

ENBDIA enables DIA and DISDIA disables it. The rest of the screen shows some test routines to demonstrate the effects of direct memory accessing.

The commands in Screen 103 set the serial port dedicated to the printer to communicate at 4800 Baud, so that we can print the listing on a slow DEC printer. FDA00C is the control register of that serial port.

Screen 104 to 107 are the commands to call firmware functions. They are defined by the defining word CAT, which expects two parameters: the command number and the number or parameters the command requires. If more than one parameter is required, they have to be given on the data stack in the reversed order as specified by the document from Digital Graphic Systems, because CAT would simple remove numbers off the stack and send them to the data register in CAT 1600. The names of commands are exactly those define by DGS for compatibility and ease of my documentation effort, which is 'Please refer to original DGS manual.'

Two test routines are shown in Screen 109 with some examples. They call the CAT functions MOVABS and DRAWABS to draw rectangles and triangles. MOVABS moves the drawing point to the x,y coordinate on the CRT display, and DRAWABS draws a line from the current drawing point to the point you specified on the stack, which is then taken as the current drawing point for subsequent drawing.

Screen 113 shows the code to build color look up tables. The word LUTCMD commands CAT to take the next 768 data words to fill the red, green, and blue color look up tables for the page 0 of the color image memory. By the words /RAMP, FLAT,

and RAMP\, we can construct quite complicated pseudo color schemes for coloring image data. One of such scheme is implemented in COMPOSITE, which displays data in the rainbow style. Small data values are in red and high data values are in blue. Data around 128 are show in green.

DIGITIZE in Screen 116 turns on the video A/D converter, and digitized signal from a video camera is shown on the CRT in real time, 30 frames per second. To freeze a frame, one has to type in the command STOP.

To Display data stored on disk, it is faster if we grab the data in a disk buffer and stuff it directly into the image memory. The code is shown in Block 126. DIA and XDIA turns the direct image access mechanism on and off. 1ROW copies 32 16 bit words from the disk buffer to the displayed image. 1BLOCK moves 1025 bytes from the disk buffer to the image memory, displayed as a 32 by 32 pixel block. READ-BLOCKS display a number of blocks, nicely arranged on the CRT display by calling 1WRITE to decide where to put the blocks.

Lastly, Screen 130 shows the commands to put text on the CRT display. CAT 1600 does not have text overlay memory like other more expensive image processors. The characters are actually drawn in the image memory and become part of the image. Therefore, one has to be careful and avoid putting text at where he has important image data.

The code shown here is a small part of the application displaying system. However, it is enough to give you a flavor on how CAT 1600 works, and to start programming it if you happen to have one. Otherwise, it's just a piece of handy reference to yet another image processor.

```
        Scr #101
(   0 ( CAT-1600 ACCESSING, CHT, 2-AUG-84)              HEX
    1 : BSWAP   ( U --- U' )
    2         0 100 U/     SWAP 100 * + ;
    3 FFFFC0. DCONSTANT DATA
    4 FFFFC2. DCONSTANT CMD
    5 FFFFC4. DCONSTANT STAT
    6 : READY ( --- F )    1 7FFF 0 DO STAT W@ 2 AND 0=
    7         IF 1- LEAVE THEN    LOOP    IF ." Cat hung!" ABORT THEN ;
    8 : INITCAT    0 STAT W!    ;
    9 : INWRD    DATA W@ BSWAP ;
   10 : OUTWRD    READY    BSWAP DATA W!    ;
   11 : OUTCMD    READY    BSWAP CMD W!    ;
   12 : STATUS    STAT W@ .    ;
   13 DECIMAL EXIT
   14 : COLOR    INITCAT   0A OUTCMD   OUTWRD ;
   15 : DEMO2   512 0 DO I COLOR 30000 WAIT 16 +LOOP    ;
C. H. TING                                              20-NOV-84

        Scr #102
    0 ( DIRECT IMAGE MEMORY ACCESS, CHT, 3-AUG-84)
    1 $600000. DCONSTANT IMAGE
    2 : ENBDIA    74 OUTCMD    ;
    3 : DISDIA    0   OUTCMD    ;
    4 EXIT
    5 VARIABLE VALUE
    6 : (IFILL)    ( LADDR COUNT --- )
    7         0 DO    2DUP VALUE @ ROT ROT W!    2 0 D+    LOOP 2DROP ;
    8 : IFILL    ( LADDR COUNT VALUE --- )
    9         VALUE !    INITCAT ENBDIA (IFILL) DISDIA    ;
   10 : STRIPE    ( LADDR --- LADDR' )
   11         BEGIN    2DUP >R DUP R> W!   2 0 D+    OVER 0= UNTIL    ;
   12 : DEMO1    ( FILL SCREEN WITH CONSECUTIVE NUMBERS )
   13         INITCAT ENBDIA    IMAGE 4 0 DO STRIPE LOOP    2DROP
   14         DISDIA    ;
   15
C. H. TING                                              20-NOV-84

        Scr #103
    0 ( PRINTER BAUD RATE CHANGE, CHT, 3-AUG-84)
    1 ( HEX        FDA00C. DCONSTANT PRCTL )
    2 HEX    4E FDA00C. B!    3D FDA00C. B!    DECIMAL
    3 EXIT
    4
    5
    6
    7
    8
    9
   10
   11
   12
   13
   14
   15
C. H. TING                                              20-NOV-84
```

Listing 10. CAT-1600 source code

```
      Scr #104
  0 ( CAT-1600 COMMANDS, CHT, 6-AUG-84)
  1 : CAT   <BUILDS   , ( ARG#) , ( CODE)
  2          DOES> 2@ SWAP   OUTCMD   ?DUP IF
  3          0 DO   OUTWRD   LOOP   THEN   ;
  4 10  1 CAT FILLSCREEN          30  2 CAT READPT
  5 23  1 CAT SETINDEX            ( 0   0 CAT DISDIA )  EXIT
  6 17  2 CAT MOVABS              31  2 CAT RDPTREL
  7 18  2 CAT MOVREL              76  2 CAT FILLREGN
  8 19  0 CAT MVPENCR             71  3 CAT NWTXTPRM
  9 24  2 CAT MVCRABS             132 1 CAT HENB
 10 25  2 CAT PVCRREL
 11 26  0 CAT MVCRPEN
 12 27  1 CAT SHOWCUR
 13 28  2 CAT DRAWPT
 14 29  2 CAT DRWPTREL
 15 20  2 CAT DRAWABS             21  2 CAT DRAWREL
C. H. TING                                        20-NOV-84

      Scr #105
  0 ( CAT COMMANDS, DT, 5-AUG-84)
  1 58 1 CAT ZOOM                 65 0 CAT WFZOOM
  2 60 3 CAT AUTOZOOM             63 1 CAT PAN
  3 62 2 CAT ROAM                 64 1 CAT SCROLL
  4 59 3 CAT ZOOMNROAM            57 0 CAT FRMGRB
  5 56 1 CAT CONTDIG              61 1 CAT EXTSYNC
  6 68 1 CAT NWDIGMSK       EXIT          80 0 CAT WFNBSY
  7 46 1 CAT NWPNTSIZE            45 2 CAT NWCSHFTRN
  8 49 1 CAT NWPENMSK             50 2 CAT NWBGND
  9 51 2 CAT NWFGND               52 1 CAT ENBKYOVR
 10 53 1 CAT ENBKYUNDR            54 1 CAT ENBNOBGND
 11 67 1 CAT NWMARKER             47 1 CAT ENBTOHOST
 12 48 1 CAT ENBFRHOST            32 1 CAT SVATRCMD
 13 33 1 CAT GTATRCMD             34 0 CAT CLRATR
 14 40 9 CAT NWOBJPARM            41 1 CAT IQOBJPARM
 15 EXIT   : DEMO3 512 0 DO I COLOR 300 WAIT 1 +LOOP   ;
C. H. TING                                        20-NOV-84

      Scr #106
  0 ( CAT COMMANDS, DT, 6/7/84 )
  1 8 2 CAT TRNSCMD               9 1 CAT DSPCHNGCMD
  2 16 0 CAT GRYLUT EXIT               69 5 CAT ENBCYCLE
  3 38 3 CAT SCRNTOLIB            36 3 CAT LIBTOSCRN
  4 37 4 CAT BFLIBTOSCRN          38 1 CAT ERAOBJ
  5 42 1 CAT NWOBORIENT           43 2 CAT ENBDRWOBJ
  6 44 2 CAT ENBTOOBJ             35 2 CAT ENBFROBJ
  7 77 1 CAT ENBFASTOBJ           95 3 CAT INCROBJ
  8 96 3 CAT SHFTOBJ              7 4 CAT GET1LUT
  9 70 1 CAT DISCYCLE             55 1 CAT ENBDISP
 10 78 0 CAT WFVBLK               79 0 CAT WFNVBLK
 11 ( 74 0 CAT ENBDIA)            3 2 CAT OUTTOPORT
 12 4 1 CAT INFROMPORT            5 2 CAT GOTOLOC
  3 6 0 CAT COMLOOP               11 0 CAT GRAYSCREEN
 14 12 0 CAT CONTINUE             13 0 CAT SHOWREG
 15
C. H. TING                                        20-NOV-84
```

Listing 10. CAT-1600 source code (cont'd)

```
      Scr #117
  0 ( CAT COMMANDS, DT, 6/7/84 )
  1 14  5 CAT BLKMOV                    15 1 CAT EXPLINE
  2 75  2 CAT IDENTIFY                  81 2 CAT NWIMGFRMT
  3 82  1 CAT NWDISPIMG                 83 0 CAT NWPENIMG
  4 84  1 CAT NWIMAGE                   85 6 CAT NWMOVIEPARM     EXIT
  5 86  0 CAT PLAYMOVIE                 87 0 CAT RCRDMOVIE
  6 88  0 CAT CONTMOVIE                 89 0 CAT STOPMOVIE
  7 90  25? CAT HISTOGRAM               118 2 CAT CONVOLVE
  8 97  1 CAT NWXFEROBJ                 98 1 CAT XFERIMG
  9 99  2 CAT ADDCONST                  100 2 CAT SUBCONST
 10 101  4 CAT MULTCONST                102 4 CAT DIVCONST
 11 103  1 CAT ANDCONST                 104 1 CAT ORCONST
 12 105  1 CAT XORCONST                 106 2 CAT SHFTCONST
 13 107  2 CAT NWOFFST                  108 2 CAT NWBOFFST
 14 109  2 CAT COPYIMG                  110 2 CAT AVGIMG
 15 111  4 CAT WSTAVGIMG                112 4 CAT DIFFIMG
C. H. TING                                      20-NOV-84

      Scr #108
  0 ( DRAW DEMO )
  1 EXIT
  2 0 FILLSCREEN
  3 : RECTANGLE    ( X1 Y1 X2 Y2 --- )
  4       2DUP MOVABS >R 2DUP SWAP DRAWABS    >R 2DUP DRAWABS
  5       DROP R> SWAP R> 2DUP DRAWABS    SWAP DROP DRAWABS    ;
  6 EXIT
  7
  8
  9
 10
 11
 12
 13
 14
 15
C. H. TING                                      20-NOV-84

      Scr #109
  0 ( RECTANGLE )
  1 EXIT
  2 : RECTANGLE    ( Y1 X1 Y2 X2 --- )
  3       2DUP MOVABS >R 2DUP SWAP DRAWABS    >R 2DUP DRAWABS
  4       DROP R> SWAP R> 2DUP DRAWABS    SWAP DROP DRAWABS    ;
  5 ( 0 0 100 200 RECTANGLE    50 50 -100 -200 RECTANGLE )
  6 : TRIANGLE  ( X Y X2 Y2 X1 Y1 --- )
  7     6 PICK   6 PICK  MOVABS DRAWABS DRAWABS  DRAWABS   ;
  8 (   0 0 45 30 11 90 TRIANGLE )
  9
 10
 11
 12
 13
 14
 15
C. H. TING                                      20-NOV-84
```

Listing 10. CAT-1600 source code (cont'd)

```
      Scr #113
  0 ( LUT, CHT, 29-AUG-84)
  1 : /RAMP   256 0 DO I 4 + 256 * I + OUTWRD 8 +LOOP   ;
  2 : FLAT   ( BYTE --- )
  3       256 0 DO   DUP 256 * OVER + OUTWRD   8 +LOOP   DROP   ;
  4 : LUTCMD   7 OUTCMD 0 OUTWRD   ;
  5 : SHOWLUT   0 0 TRNSCMD   0 DSPCHNGCMD   ;
  6 : RAMP\   0 255 DO I 4 - 256 * I + OUTWRD -8 +LOOP   ;
  7 : COMPOSITE   LUTCMD
  8       ( RED) /RAMP 255 FLAT RAMP\ 0 FLAT
  9       ( GRN) 0 FLAT /RAMP 255 FLAT RAMP\
 10       ( BLU) 0 FLAT 0 FLAT /RAMP 255 FLAT
 11       SHOWLUT   ;
 12 COMPOSITE
 13
 14
 15
C. H. TING                                          20-NOV-84

      Scr #114
  0 ( BITPAD INPUT, CHT, 9-AUG-84)    EXIT
  1 $FD4030. DCONSTANT BPDATA
  2 $FD4032. DCONSTANT BPSTATUS
  3 $FD4034. DCONSTANT BPMODE
  4 $FD4036. DCONSTANT BPCMD
  5 $27 BPCMD B!
  6 : BP@   BEGIN BPSTATUS B@ 2 AND UNTIL BPDATA B@   ;
  7 : BP!   BEGIN BPSTATUS B@ 1 AND UNTIL BPDATA B!   ;
  8 : XY@   BEGIN BP@ 6E = UNTIL   4 0 DO BP@ LOOP   ;
  9 : >XY   ( N1 N2 --- X )
 10       63 AND 64 *   SWAP 63 AND +   ;
 11 : BITPAD ( --- Y X )
 12       XY@   >XY >R   >XY R> SWAP   ;
 13 : XYTEST   BEGIN CR BITPAD 5 .R 5 .R ESC UNTIL   ;
 14 : BPWAIT   ( ___ VALUE )   BEGIN   BP@ DUP 64 AND   0=
 15       WHILE DROP REPEAT   ;
C. H. TING                                          20-NOV-84

      Scr #115
  0 ( CURSOR CONTROL BY BITPAD, CHT, 9-AUG-84) EXIT
  1 : GETXY   ( --- Y X )
  2       BITPAD 2/ 2/ 256 -   SWAP 2/ 2/ 256 -   SWAP   ;
  3 : CURSOR   ( Y X --- )
  4       MVCRABS   1 SHOWCUR   ;
  5 : CURTEST BEGIN GETXY CURSOR   ESC UNTIL   ;
  6 : DRAWING BEGIN   GETXY 2DUP CURSOR DRAWABS   ESC UNTIL ;
  7
  8
  9
 10
 11 EXIT
 12
 13
 14
 15
C. H. TING                                          20-NOV-84
```

Listing 10. CAT-1600 source code (cont'd)

```
      Scr #118
 0 (  FLASH DIGITIZER, CHT, 9-AUG-84)
 1 : DIGITIZE
 2         1 EXTSYNC    0 NNDIGMSK
 3         1 CONTDIG
 4       ;
 5 : STOP    0 CONTDIG    0 EXTSYNC    ;
 6
 7
 8
 9
10
11
12
13
14
15


      Scr #126
 0 (  IMAGE WRITE, CHT, 28-AUG-84)
 1 : DIA  ENBDIA STAT W@ DROP  ;
 2 : XDIA   0 OUTCMD   STAT W@ 1 AND IF DATA W@ DROP THEN ;
 3 : 1ROW ( LADDR ADDR --- )
 4         32 OVER + SWAP DO   2DUP I @ ROT ROT W!
 5         2 0 D+   2 +LOOP   2DROP   ;
 6 : 1BLOCK ( LADDR ADDR --- )
 7         32 0 DO  2DUP R@ 1ROW   512 0 D+ R> 32 +   LOOP
 8         DROP 2DROP   ;
 9 : 1WRITE  ( N BLOCK --- , WRITE TO NTH FRAME)
10         BLOCK >R   DUP 15 AND 32 * 0 ROT
11         240 AND 1024 UM* D+   IMAGE D+   R> 1BLOCK   ;
12 : READ-BLOCKS   ( BLOCK# N --- )
13         DIA   0 DO I OVER I + 1WRITE ESC IF LEAVE THEN LOOP
14         XDIA   DROP   ;
15


      Scr #130
 0 (  TEXT DISPLAY, CHT, 26-AUG-84)
 1 : DRWTXT   ( ADDR COUNT Y X --- )
 2         72 OUTCMD   OUTWRD OUTWRD
 3         OVER + SWAP DO   I @ READY DATA W!   2 +LOOP
 4         0 OUTWRD   ;
 5 : DEMO8   ( BLOCK --- )
 6         BLOCK 1024 0 DO
 7         I OVER +   64 200  I 4 / - -200 DRWTXT
 8         64 +LOOP   DROP   ;
 9
10
11
12
13
14
15
```

Listing 10. CAT-1600 source code (cont'd)