168
78

To the Graduate Council:

I am submitting herewith a thesis written by James Russell Younkin entitled "A Microcomputer-Based Electronic Radiographic Inspection System Using Digital Image Processing Techniques." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Donald W. Bouldin

Donald W. Bouldin, Major Professor

We have read this thesis
and recommend its acceptance:

Robert W. Rochelle

Dejarnette Rogers

Accepted for the Council:

Lewis Winkel

Vice Provost
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirments for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature James Russell Younkin

Date June 1986

A MICROCOMPUTER-BASED ELECTRONIC RADIOGRAPHIC INSPECTION SYSTEM USING DIGITAL IMAGE PROCESSING TECHNIQUES

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

James Russell Younkin

June 1986

ACKNOWLEDGEMENTS

The author is very grateful for the invaluable guidance, and helpful advice of Dr. Dragana Brzakovich, Dr. Donald W. Bouldin, and Dr. Robert Rochelle. An enormous amount of useful information has been obtained while researching this material.

Additional special recognition is given to my family for their support, understanding, and encouragement throughout my education.

ABSTRACT

The purpose of this task was to provide the hardware and software requisite to implement an electronic radiographic workstation that would allow the suitability of both image processing techniques and imaging components to be evaluated as candidate solutions for Non-destructive evaluation requirements in an industrial environment. The workstation is based around the LSI-11/73 microcomputer and its Q-bus architecture and the single-user RT-11 operating system. An image processor and array processor have been integrated into the host computer system. Specific software was developed to provide a reasonable performance, low cost, workstation. Workstation features include expandable mass storage, hard copy image output, 4 Mbyte of accessible array storage on the host computer, direct frame buffer access to data by the array processor, continuous histogram display, and frame buffer arithmetic operations. As a culmination to this study, the workstation and image processing techniques are used as an automated solution to the problem of detecting and measuring foreign inclusions within a material before it is mechanically processed. The steps involved toward automating this inspection include the acquisition of the image, preprocessing the image, thresholding it with a global threshold, and locating and measuring the impurities. The preprocessing step is composed of a non-linear filtering algorithm that reduces the noise in the image and preserves the defect's edge information and a polynomial-fit-subtraction technique to eliminate a slowly varying background gradient present in

the acquired images. These two techniques put the image data into a more suitable form to permit further automated processing. The effects of the processing step allows the area of 0.020 inch diameter test spheres to be measured within 2.5% by computer measurement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The Non-destructive Evaluation (NDE) field is actively involved with electronic radiography for the purpose of advancing methods of inspection that do not destroy or damage an item in order to evaluate its strength and/or its quality [1,2]. The increased interest in electronic radiography has stemmed from the improvements in the performance of the X-ray detectors [1] and the improved capabilities of the digital image processing equipment. The prowess of currently available image processing hardware encompasses electronic accumulation or integration of radiographic data for weighted or slide averaging, video-rate frame arithmetic and boolean operations, digital image processing operations using digital convolution, and frame operations including histogram generation. Other advantages of electronic radiography making this method of inspection attractive are the considerably lower cost due to the increasing expense involved with film inspection, increased inspection throughput, the ability to perform the inspections on every production item, and the ability to detect defects by moving the object being inspected through the X-ray beam [1].

Electronic radiographic systems have been assembled that rival the image quality obtained from conventional film radiography [1,2]. Similar to film radiography, the radiation generated from the X-ray source probes the item being inspected. In electronic radiographic imaging systems however, the two-dimensional modulated radiation inherently produced by the density and thickness variation of the item

under test is not recorded on film but is detected by an X-ray-to-visible-light converter. A camera system coupled to the converter permits the visible image to be converted to an analog signal which is fed to a digital image processor for digitization. The digitized image data is then processed or manipulated to either enhance the image for human obervance or measure and report features in the image automatically [1].

A portion of the impetus for acquiring radiographs electronically is derived from economics. A substantial financial benefit can be gained when electronic sensors and digital storage devices are used to replace conventional film radiography. Calculations have been performed offering a projected $4,500,000 savings over a 10 year period for a 13 room Radiology department performing 64,000 procedures a year [3]. Another force driving the implementation and performance improvement of electronic radiography is the ability to apply digital image processing techniques to overcome the deficiencies presented in these images and provide a result that is more suitable than the original image for both human observance and subsequent machine interpretation. The degradation of the images, or deficiencies within, characteristically have low resolution, low signal-to-noise ratio (SNR), low contrast, lack of smoothness of illumination, and geometric unsharpness. Digital image processing not only offers methods that can greatly improve the degradations of these images but also offers techniques that can enhance features in these images and allow implementation of automated interpretation or inspection of those images.

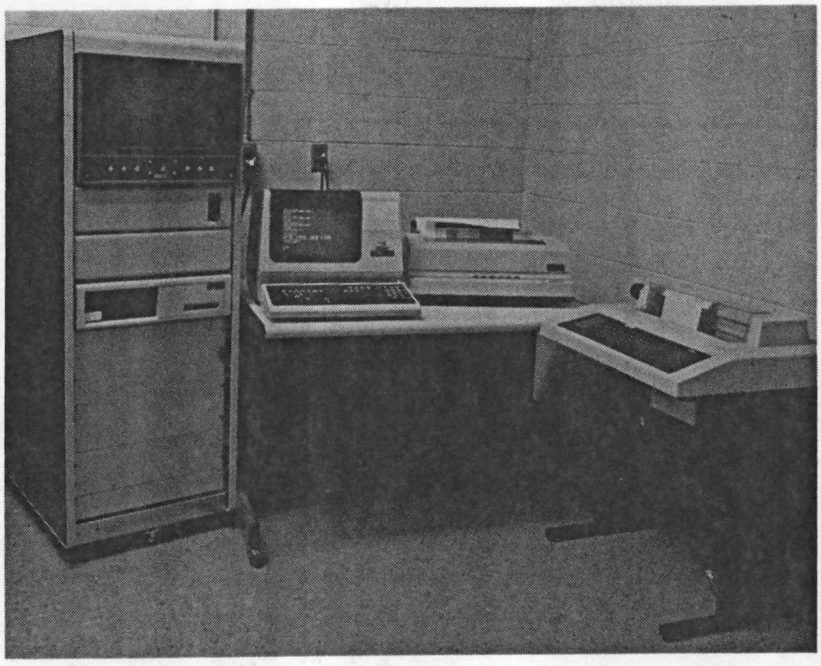
This thesis is aimed toward providing an understanding of the hardware and software aspects of implementing this specific electronic radiographic system utilizing digital image processing as a resource to simplify and improve inspection automation. The description of the developed Electronic Radiographic Inspection System (ERIS), the integrated components, and the system integration philosophy is detailed in Chapter 2. Chapter 3 provides an itemized discussion of the software that was developed specifically for the hardware composition of ERIS. Chapters 4 and 5 discuss image processing techniques and their hardware/software implementation on ERIS and the roles that these techniques play in improving image quality of electronically acquired radiographs. Chapter 4 explains noise reduction methods while Chapter 5 explains a method to obtain flat fields from images containing strong unwanted gradients. Chapter 6 details the use of ERIS in performing a simulated inspection task.

CHAPTER 2

THE ELECTRONIC RADIOGRAPHIC IMAGE PROCESSING WORKSTATION

The components of the Electronic Radiographic Inspection System, ERIS, have been integrated to create the single-user image processing workstation pictured in Figure 1. The system is centered around Digital Equipment Corporation's LSI-11/73 microcomputer and its Q-Bus bus structure. Standard off-the-shelf hardware was purchased to implement the system. An itemized cost list of the components comprising the system is contained in Appendix A. The system's hardware consists of the central processing unit (CPU), memory, peripheral interfaces, array processor, and image processor. The power requirements of this hardware (Appendix B) permits a single enclosure/power supply to house the system's processing section. The system will automatically power up and bootstrap itself when power is applied.

Before discussing the components of the system, the reasoning leading to the selection of the host computer, the LSI-11/73, should be elucidated. Due to the extent at which image processing is being employed in the industrial environment, a system integration and implementation strategy must be cultivated to insure connectivity, expandability, flexibility, and growth alternatives for those image processing systems that will be placed in that environment. These guidelines were established in order to place constraints on the selection of the components of the workstation. Cost of the system components must also be considered. If image processing systems are to continue to expand into the industrial environment, their cost/performance effectiveness must be attractive.



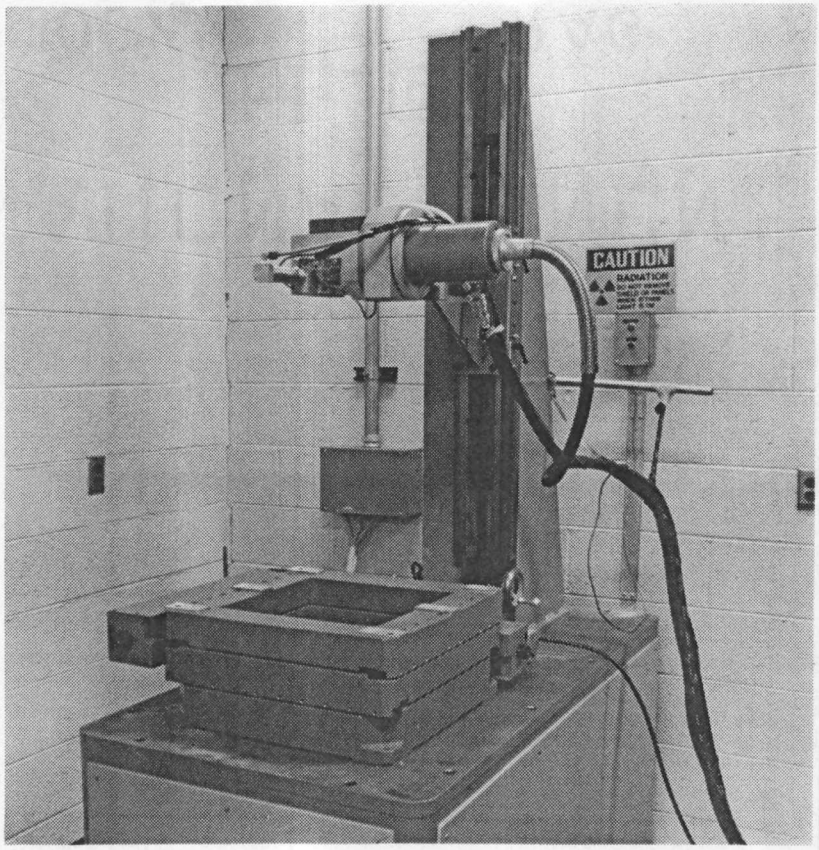


Figure 1. The Electronic Radiographic Inspection System (ERIS).

The image processor/host connection is a vital link of the workstation. To attain high performance and the ability for automation, the image processor must have the capability to accept directives directly from the host computer bus. Connectivity not only refers to the ability to easily integrate the image processor and host computer that will be required to implement the workstation but also refers to the ability to connect to the outside world. The workstation needs the network capability that allows information transfer among the systems in an industry. This requirement is very important if Computer Integrated Manufacturing (CIM) is being implemented in the environment.

Many vendors of image processing equipment offer flexible systems that allow the implementation of a variety of configurations of hardware. These systems range from simple frame digitizers to complete systems containing many frame buffers, video rate arithmetic processing units, histogram/feature extraction modules, software assisted Discrete Fast Fourier Transform hardware, and even Real-Time convolvers [4]. To reduce system cost and complexity, the image processing system integrator selects the minimum set of image processor modules to configure a system capable of performing the image processing task. If the task changes, modules can easily be removed or new ones inserted. The ability to expand or add peripheral equipment to the system as technology and vendors provide and support them is also required. The electronic radiography of objects require the analysis of data arrays or files ranging in size up to .25 Mwords (512X512X16 bit image). Analysis of these quantities of data require rapid processing and large storage capacity. As faster, larger disk drives become available, the work station should be able to accommodate them.

The selected host is of the PDP-11 family of 16-bit processors. Due to the familiarity with the bus structure, the variety of peripherals available, the excellent cost/performance effectiveness, and the variety of image processing equipment that can be connected to it, the work station is based around the LSI-11/73 microcomputer and its Q-bus architecture. (Modular, plug-compatible image processing equipment for the Multi-Bus and VMEBus [4] are also available.) To assist the LSI-11/73 when performing computations, it has been "turbo-charged" with a Q-bus plug-compatible array processor. The image processor that has been selected for inclusion into the workstation is a set of building block modules that are also Q-bus plug-compatible. This allows both the host and the image processor to be configured into a single package. Selecting the LSI-11/73 as the host allows the Microvax II, with its fast floating point accelerator, 16Mbytes of memory on its Local Memory Interconnect (LMI), and its Q22-Bus map high speed data link, to be the growth alternative for the LSI-11/73.

The approach taken in implementing the workstation has been to provide all the available tools to the user to allow subsets of the work station to be extracted to solve a particular task in the field. As an example, the solution of an image processing task turns out to be very complex and CPU intensive. It is decided to place a frame digitizer in the industrial environment linked to a main processing system. After the frame is digitized, the data is transferred over the network to a main processing system for processing and mensuration. The workstation consists of the full complement of image processor building blocks that

allows an image processing problem to be solved initially before placing a minimal system in the field.

The remainder of this chapter will discuss the components contained in the implemented Electronic Radiographic Inspection System. Figure 2 details the component interconnection scheme. This figure should be referenced during the discussion that follows. It is desirable to traverse the system configuration from X-ray generation to the stored processed image. Each sub-system will be described.

Imaging

The Imaging subsystem configuration, consisting of the components required to obtain a video image of an object under test, is considered transmission radiography. An X-ray generator providing the probing illumination and an X-ray detector converting the X-ray intensity into a video signal are the components of this sub-system. The X-ray source or generator being used is a GE XRD-6, 75 kV unit. This generator is considered archaic yet was readily available for use as a component of this system.

The simplest form of transmission radiography, shown schematically in Figure 3, consists of an X-ray source, the test object being radiographed, and an image detector on which the shadow of the test object is projected. X-rays are produced by focusing a beam of high energy electrons onto a small focal spot on the anode. The number of X-rays emitted from the anode or target, the intensity of the X-ray beam, is a function of the cathode current and the anode voltage. In the absence of a test object, the X-ray intensity or dose reaching the image

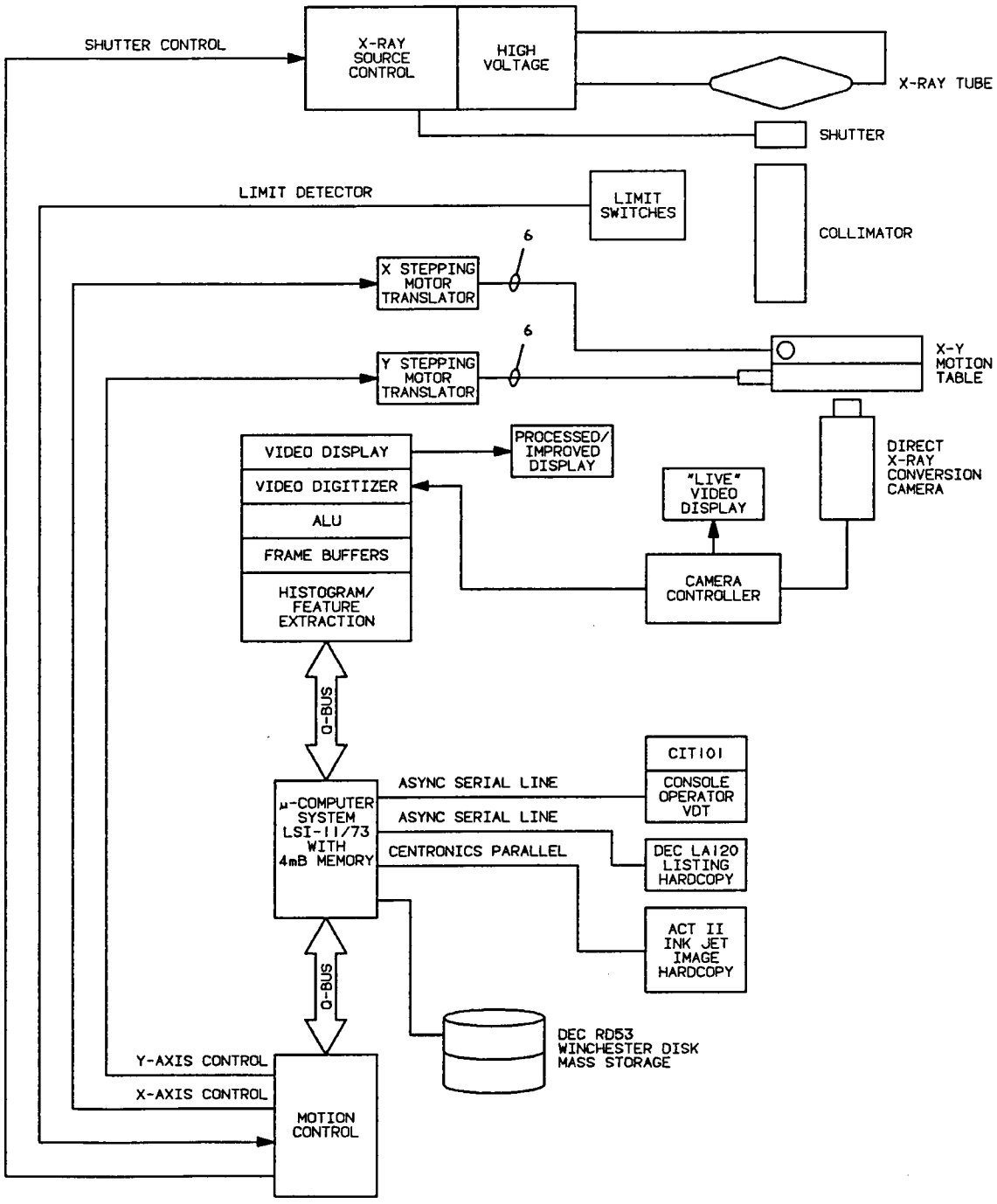


Figure 2. ERIS Interconnection Diagram.

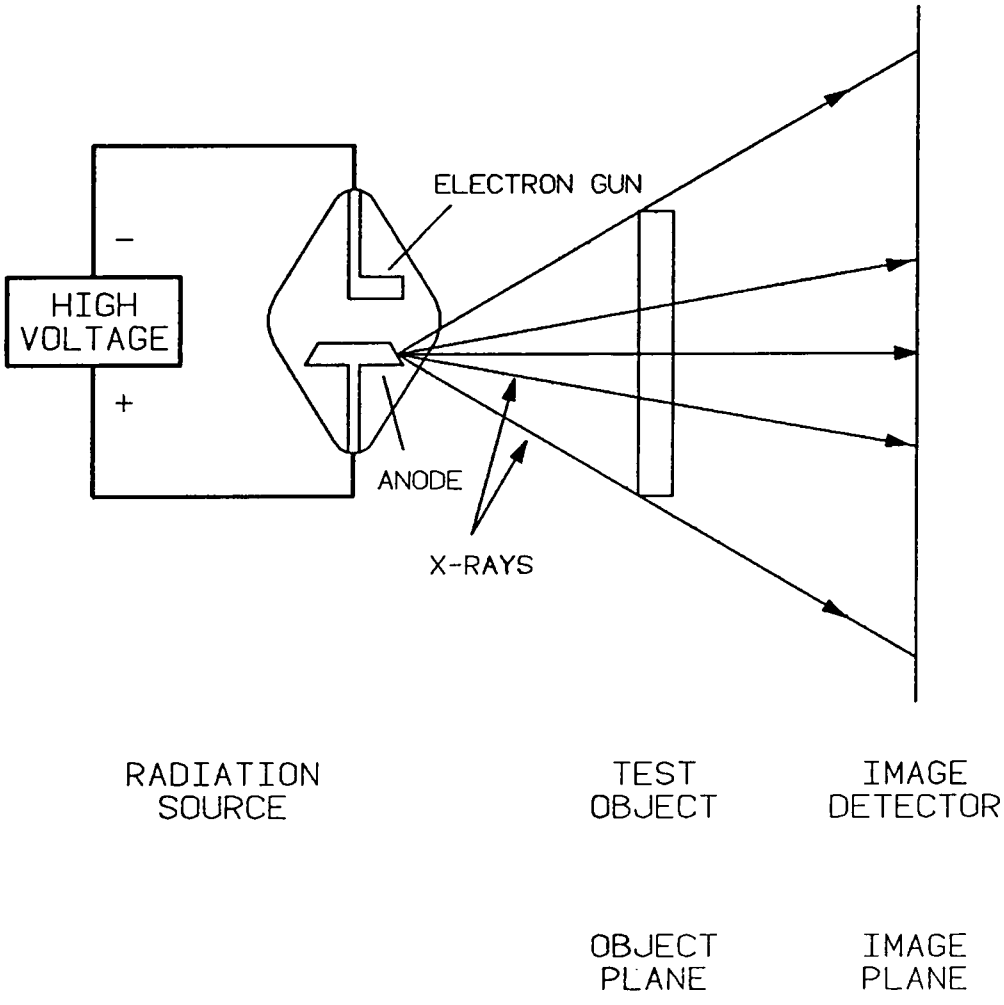


Figure 3. Basic transmission radiographic imaging configuration.

plane is only reduced by the reciprocal of the square of the distance from the source. When the radiation penetrates a test object, the two-dimensional intensity distribution reaching the image detector is also altered by the mechanisms of absorbtion and scattering. The physical basis for the attenuation of an X-ray penetrating the test object is intrinsic to the material's composition and thickness and the X-ray energy. The "effective" thickness of a material refers to either the actual thickness of the material or a perturbation in the composition characteristic of a void or foreign inclusion. Further treatment of the principles of radiography can be found in References 5 and 6.

A critical component in an electronic radiographic imaging system is the X-ray to light/electronic image converter [1]. This component consists of two parts, a component to convert X-rays to visible light and a component to convert the light image into a video signal capable of being digitized by an image processor and displayed on a video monitor. Two X-ray detectors have been included into this system. For work requiring high resolution, a direct X-ray conversion video camera can be connected. This camera contains a 1 inch X-ray-sensitive vidicon tube that directly converts the X-ray flux striking a Beryllium target window into an electronic signal. This camera is to be used in the low to mid energy range ($\leq$ 150 kV) and requires a high dose rate (50 to 500 R/min.). The sensing area of this vidicon is 9.5 X 12.5 mm, lending itself to a .018 mm/per line resolution for a standard 525 line scan-rate or .072 mm to define an object in 4 lines. The X-ray-sensitive vidicon is good for detecting small objects. X-ray penetrameter

sensitivities of 2% have been obtained [5]. For larger field requirements a direct X-ray image intensifier can be installed. This unit has the X-ray screen, photocathode, coupling optics, and video camera combined into a single package. The input window of this device is 12 inches.

Test Object Transport

A stepping motor motion control system has been included as an integral part of the system. Movement of the test object allows the objects to be aligned properly and also provides the ability to move the object under test through the X-ray beam. The stepping motors are computer controlled to allow automation of inspection tasks. Although two axes are currently attached, the software and computer hardware exists to control five ranges of independent, concurrent motion. The resolution of the X-axis and Y-axis and the range of travel are .0005 inch and 14 inches respectively.

Image Processor

Due to the large number of pixels in a digital image, an image processor (IP) capable of digitizing, processing, and displaying those digital images is another key component of the workstation. An image processor allows the processing performance required for interactive processing of images to be achieved [7]. The selected image processing system (IP-512) is from Imaging Technology, Incorporated. This modular image processing unit consists of a set of Q-bus compatible boards that are directly plugged into the host microcomputer backplane and that can be directly accessed by the microcomputer through its I/O page. The

boards contained in the ERIS configuration consist of the AP-512 Analog Processor module, the ALU-512 Arithmetic Logic Unit, and four FB-512 frame buffers configured as two 512X512X8 buffers and one 512X512X16 buffer. The system also includes the HF-512 Histogram generator/Feature extraction module. The software package ITEX-Q is also available to provide communication between the host computer and the IP. To provide adequate display information, an RGB monitor has been obtained to display pseudo-color intensity slicing, and two monochrome monitors to allow a parallel viewing of the original image and of the enhanced image.

The image processor, acting as a high-speed auxiliary processor [7], is directed by the host computer through control, status, and data registers existing in the I/O page. The ideal image processing task exists when all of the processing can be performed by the image processor leaving the host computer as the control sequencer for the image processor and as the disk server to provide mass storage for the image data. For many image processing operations this is the case. The processing can be performed directly on the images stored in the IP's frame buffer memory with a minimum of host access of the actual image data. The algorithms performed on an image processor can be categorized as frame processes, group processes, or point processes [4]. Point processes involve operating on the intensity values of each pixel in an image. The capability to perform arithmetic, boolean and logical functions between two images or an image and a constant exists. Real-time point processing is easily performed on the IP by translating the stored intensity values in a frame buffer through the IP's output lookup

tables prior to being displayed. The contents of the frame buffer are not immediately changed. This capability allows various intensity mapping functions to be evaluated interactively [7]. Once the display results are deemed desirable, the values in the frame buffer can then be mapped. The active window display area allows group processes such as digital convolution to be performed. The neighborhood of a pixel is easily accessed by allowing a frame buffer to be offset relative to another frame buffer in both the x and y directions. Frame processes allow statistics to be acquired with regard to the intensity variation in an image without the host computer accessing all of the pixels in a frame buffer. Existing hardware functions provide the location and number of pixels meeting established intensity values and the histogram of grey level occurrence. Another useful feature inherent to the IP is the ability to protect pixels with a binary valued image and the ability to protect data planes. Pixel protection is used to define a Region of Interest (ROI) where processing is allowed to be performed. The region not to be accessed is protected from change.

Microcomputer

The LSI-11/73 based microcomputer contains 4Mbytes of random access memory (RAM). The card cage location map for the ERIS is illustrated in Figure 4 to demonstrate the relative position of each circuit board installed in the Q-bus backplane. A high density RAM memory board is used to provide system memory. The single quad-size board contains the full 4 Mbytes of RAM and provides parity error support. A dual port asynchronous serial line unit with boot PROM,

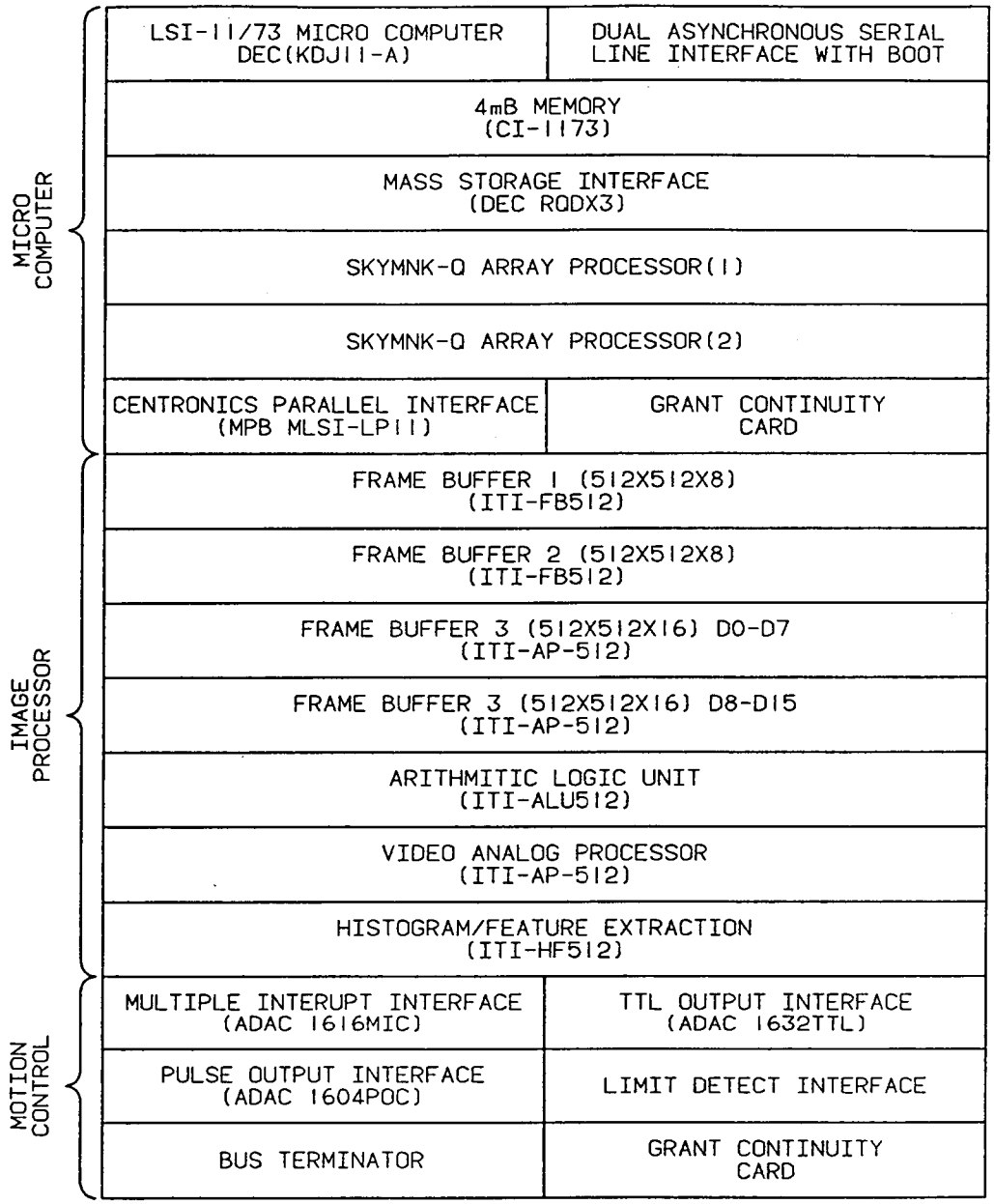


Figure 4. System card cage location map.

parallel unit, and mass storage interface constitute the remainder of the microcomputer.

When processing images, the host computer is required to perform the calculations that can not be performed on the image processor. To lessen the computational load on the host and thus increase system throughput, an array processor (AP) has been incorporated as an integral part of the workstation's computer subsystem for rapid advanced signal analysis which includes the calculation of 512-point Fast Fourier Transform in 27 mS. Array processors are efficient at performing high-speed operations on sets of data such as video data lines. An array processor incorporated into a computer system has been shown to reduce program execution by more than an order of magnitude [8]. The AP has a host specific interface to provide Direct Memory Access (DMA) I/O between the host computer and its control processor which is a pipelined arithmetic unit. DMA protocol allows data transfers to be initiated and controlled by the AP with a minimum of CPU intervention. Since the CPU is not involved, the DMA mode of operation can perform data rate transfers of up to 500 KWords per second. Speed is not only attained by implementing DMA data transfer protocol and pipelined architecture, but also by the ability to perform overlapped and parallel processing. The overlapped nature of the AP allows DMA input transfers to be performed to obtain data for the next arithmetic operation and DMA output transfers from the previous operation concurrently during the execution of the current operation. Since the host computer and array processor operate asynchronously, the independent nature of the AP permits

distributive processing techniques to be used to assure both remain active [8].

The array processor consists of a two quad-size board set directly plugged into the Q-Bus system backplane. The cards are interconnected via a 34 pin ribbon cable. A set of Fortran and/or Macro callable subroutines are available to the program developer in the vendor supplied object module MNKLIB. However, greater performance from the array processor can be obtained through the direct access routines contained in MNKLIB.

Peripherals

The computer peripherals consist of the console, printers, and mass storage devices. System mass storage is currently provided in the form of a 71Mbyte 5.25" Winchester disk. The Winchester is used primarily for the operating system, image file storage, and image processing programs. Other types of storage devices will be incorporated as a system feature in the future. The console terminal is used for command initiation and program development. A printer is available to obtain software listings and inspection reports. A drop-on-demand color ink-jet printer has also been included as a system peripheral to provide permanent outputs for comparing image data sets and for reports. From the printer's three primary ink colors, yellow, magenta , and cyan, the printer can print up to 125 different color shades. Using drop-on-demand ink-jet technology, print speeds are achieved up to 5.6 inches per minute (full width). Centronics compatible parallel communications protocol is used for interfacing with the computer. The printer may be

programmed from the computer through the use of special control codes. Various operator controls are provided. Commonly used controls are available on the front panel; additional controls may be accessed by opening the front cover. In addition to self-test options, the printer has built-in diagnostics for troubleshooting programming and communications problems. The diagnostics include hexadecimal dump mode, checksum verification, and ink pump checks. The hexadecimal dump mode is a useful diagnostic when developing software to drive the ink jet printer. When this mode of operation is selected, the hexadecimal code for each of the characters received is printed without the usual control character interpretation.

Operating System

The DEC RT-11, version 5.1 single-user operating system was selected to run on this system. This operating system is a real-time, disk based operating system designed for interactive program development and on-line applications. RT-11 efficiently uses system resources to minimize system requirements on the CPU and on the mass storage device while maximizing system throughput. The ease of use of RT-11 is due in part to its simplicity. Some of the features provided by RT-11 are its contiguous file structure, flexible real-time I/O, low system overhead, and ease of generation and expansion. The system programs that are normally supplied with the system are very powerful. Some of the programs that are available are LINK, KED, MACRO, RESOURC, LIBR, DUP, PIP, and DIR. The RT-11 operating system also has a variety of useful utility programs such as ODT, VDT, DUMP, SRCCOM, BINCOM, FORMAT, PATCH,

SIPP, PAT, and SLP. The Fortran IV high level programming language has also been included with the system. Two newly added RT-11 operating features have been installed, tested, and found to be very useful both in the industrial environment and in the program development environment. The first of these new software components is the Transparent Spooling Utility (SPOOL). SPOOL runs as a foreground job and thus allows work to be continued in the background while files are spooled to the output device. When running, SPOOL automatically intercepts all data directed to the designated output device. The commands required to load the output device's handler, assign the SP pseudohandler the name of the output device logical name, and run SPOOL are typically contained in the start-up indirect command file. The other new software component is the Virtual Terminal Communication Package (VTCOM). VTCOM lets the stand alone ERIS system appear as a local terminal to other host systems. This utility not only lets the stand alone system appear as a terminal, but it also allows the transfer of ASCII files between systems. Binary files may also be transferred with the use of the program TRANSF. VTCOM can run under both Foreground/Background monitor (FB) and Single Job monitor with timer support. VTCOM uses the special handler XL.SYS to provide the proper software connection to the hardware serial port being used as the host connection. To run the utility VTCOM, under the single job monitor, the user types "RUN VTCOM.REL". Running VTCOM under the FB monitor provides a very useful and efficient file transfer environment. Operating VTCOM in the FB environment allows access to both systems without exiting VTCOM by switching between the foreground and background. Command files

that initiate the VTCOM environment and the SPOOL feature and information regarding special line printer support for the operating system is contained in Appendix C.

CHAPTER 3

SPECIAL SOFTWARE

Several of the generic image processing routines that are standard to any image processing system are listed in Appendix D and have been written for inclusion into the ERIS software package. To achieve better performance from these image processing algorithms and to be able to fully use the resources of the microcomputer system and the features of the image processing hardware, system dependent software has been written.

Frame Buffer Data Storage/Retrieval

The initial assignment demanded of the host computer system was the ability to store and retrieve images to and from mass storage. The software module IPFIO (Image Processor Frame Input Output) has been written in standard PDP-11 Macro assembly language to allow frame buffer data to be directly read to or written from the host computers mass storage units. This software explicitly uses the operating system programmed requests and microcomputer resources. A 512X512 frame buffer requires four seconds for a complete data transfer. Currently no data compression coding is used to reduce the data file size. Data compression schemes offering compression factors from 2 to 3 without degredation of image quality are available and being considered to reduce image data file sizes on the ERIS [9].

Dynamic Virtual Array Storage Software

For some image processing applications it is necessary to place

image data into the host's memory. Performing the digital FFT and being able to store intermediate processing results without having to access mass storage are two applications where it would be advantageous to have the ability to consume a large block of host memory. Since the LSI-11 family of processors cannot directly address memory above 64 Kbytes as 512X512 arrays, a dynamic memory channel allocation scheme has been developed that implements the addressing of the virtual memory of the system with the assistance of the Memory Management Unit (MMU). This method solves the problem of storing and accessing the large image memory requirements above the physical address space. Mapping or translating the 16 bit address of the LSI-11 processors into the 4Mbyte physical memory is accomplished with the use of bits in the processor status word and the registers of the MMU located on the CPU board. The Processor itself cannot directly address the 4mbytes of memory. The Dynamic Virtual Array Storage Software (DVASS) accomplishes the mapping by manipulating the MMU registers and using the following basic principles.

The technique of accessing virtual memory is depicted in Figure 5. The MMU uses two sets of 8 32-bit relocation or base address registers, known as Active Page Registers (APRs), combined with a 16-bit virtual address to form a 22-bit physical address. The APR set is selected by bits in the Processor Status Word. The DVASS sets bits in the PSW so that the system APRs remain as kernel, but the user APR set is set to previous. The MMU is turned on and 22-bit mode selected through the MMU status registers. The page address register is then loaded with the base address in 64 bytes of the memory to be used. The instructions

```
        TO CLEAR A 32K-WORD BLOCK OF MEMORY FROM THE 64K-WORD BOUNDARY TO THE
           96K-WORD BOUNDARY
        LOAD USER APR VALUES
        SET PREVIOUS  MODE AS USER IN PROCESSOR STATUS WORD
        ENABLE RELOCATION IN MEMORY MANAGEMENT STATUS REGESTER
        ENABLE 22 BIT ADDRESSING IN MEMORY MANAGEMENT STATUS REGISTER
          .
          .
          .
        CLR R0          ;POINTER TO APR BEING USED AND LOCATION WITHIN APR WINDOW
        CLR R1          ;VALUE TO WRITE TO MEMORY
1$:     MOV R1,-(SP)    ;PUSH VALUE TO PUT INTO MEMORY ON STACK
        MTPI (R0)       ;PUT VALUE ON STACK INTO PREVIOUS MODE SPACE INDICATED
                        ;BY R0 POP STACK, AND INCREMENT ADDRESS BY 2
        TST R0          ;SEE IF DONE
        BNE 1$          ;LOOP IF NOT
```

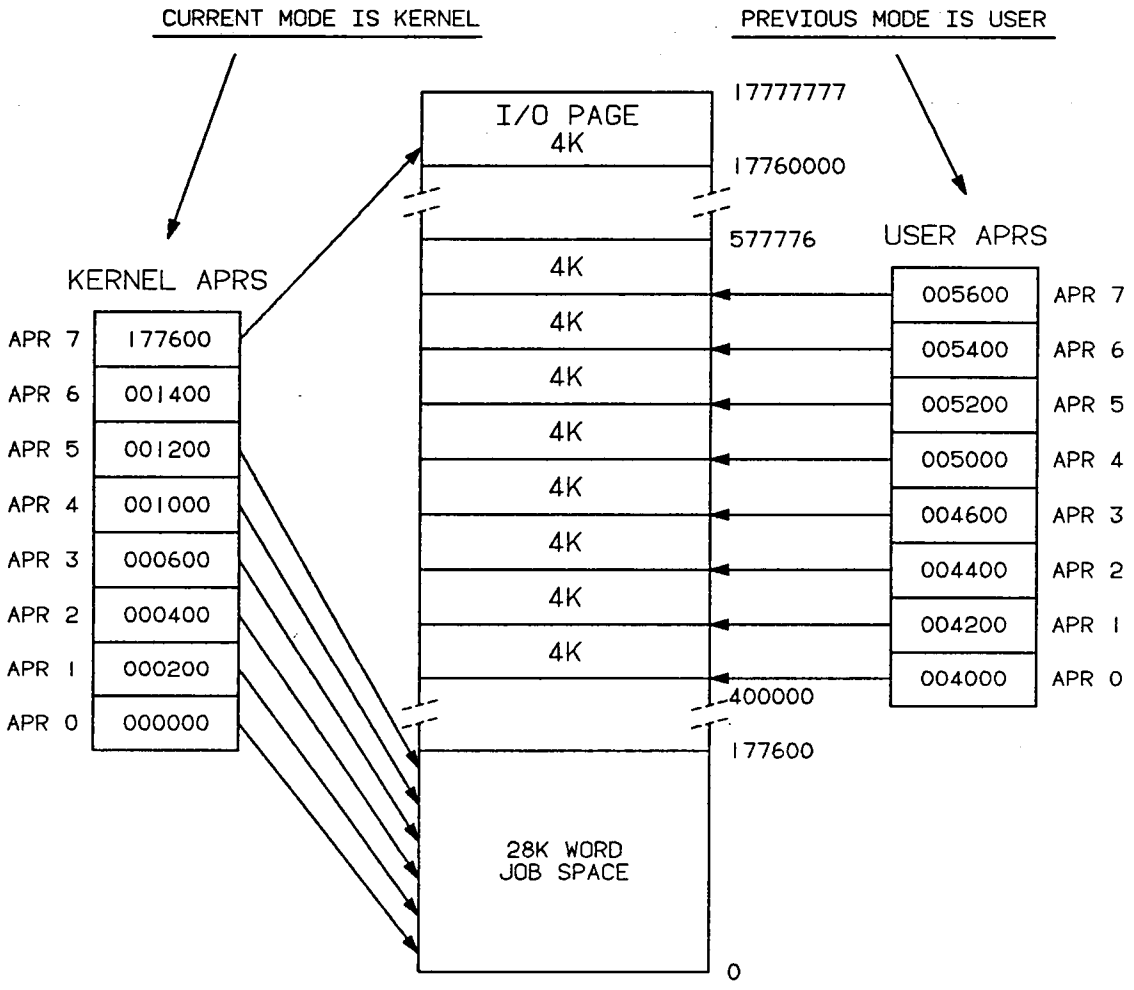


Figure 5. Accessing virtual memory on an LSI-11 CPU.

MFPI and MTPI are used with the offset within the page to select the desired memory location.

In the DVASS scheme, a channel number and size is defined by the application software by specifying a channel number and element type (byte, integer, real, complex). From this information, the virtual memory software dynamically allocates the required space for a 512X512 array for the specific element type. Table 1 discloses the mixture of possible array types that can coexist in the virtual array space. With 4 Mbytes of memory, fifteen 512X512X8 copies of frame buffers can exist. The software that manages the virtual memory array storage also allows integer, real, and complex 512X512 element arrays to be accessed. The memory can be partitioned to contain a variety of array types. This software can be easily adapted for other uses and array sizes. It must be kept in mind that the RT-11 operating system and I/O page require 64 Kbytes of the total memory of the system. The software has been written to allow access to the image data stored in computer memory on a row, column, or element basis.

The routine FFTFIL has been included in DVASS to improve the FFT computation performance by utilizing the conjugate symmetry property of the origin centered FFT. FFTFIL generates the right half of the FFT from the left half without performing the FFT on the columns of the right half. When computing the FFT of an image, the data is read directly from the frame buffer by the array processor as single lines. After the array processor performs the FFT on the "row," a DMA transfer is performed to write the results from the array processor directly to an allocated storage block in virtual memory. To process the "columns,"

Table 1

512X512 Virtual Array Allocation in 4 Mbytes

Total # of Arrays	Array Type			
	Complex	Real	Integer	Byte
4	1	1	1	1
5	1	-	3	1
6	1	-	2	3
7	1	-	1	5
8	1	-	-	7
5		3	1	1
6		3	-	3
6		2	3	1
7		2	2	3
8		2	1	5
9		2	-	7
7		1	5	1
8		1	4	3
9		1	3	5
10		1	2	7
11		1	1	9
12		1	-	11
8			7	1
9			6	3
10			5	5
11			4	7
12			3	9
13			2	11
14			1	13
15				15

the data in virtual memory is accessed directly by the AP as columns and the FFT is reperformed. A complete list of DVASS functions are located in Appendix E. Table 2 has been included to depict the relative speed of transferring data among system resources by the various mechanisms offered by the host computer system.

Histogram/Feature Extraction

The histogram/feature extraction module (H/F) has proved to be invaluable in both increasing the speed of the image processor software and aiding X-ray operational setup for improved images. A 256 element histogram can be obtained in 1/15 second on the data presented by the selected video source. Acquiring and displaying the histogram continuously allows the X-ray operator to determine how well the X-ray intensity distribution covers the digitizer grey levels. While viewing the histogram display, the operator can adjust the beam current and voltage to obtain the optimum dynamic range and contrast. The feature extraction operation provides the X-Y coordinate pairs for 4096 pixels in the video source in 1/30 second. Four lookup tables (LUT) exist on the H/F module. Two are used by the histogram operation for translating intensity values prior to histogramming. The other two are used to select intensity values that are deemed to be located as features. Several support functions have been written that simplify the implementation of the general purpose image processing routines. These functions are contained in the software module HSFX. The abundant use of the histogram/feature extraction module is considered to be a vital and efficient method of implementing the software as fast as possible.

Table 2

Byte Frame Buffer Transfer Times
512X512 Image

Method	Time In Seconds
Copy Through ALU	1/30
Through FB data registers	2.1
To Disk	4.1
To virtual memory	2.3

This software has been written as Fortran callable assembly language routines. The H/F module is directly connected to the video bus and is continually operating on the data. Routines that return the minimum and maximum intensity values of an image and find the occurrence and location of pixels of selected intensity values are typical of the type of functions that can be performed. The full compliment of HSFX subroutines and DVASS subroutines are contained in Appendix E.

Plot Image

The program ACTPLT has been created to allow color copies of Frame buffer images or image data files to be obtained from the ACT II color ink jet printer. Examples showing the palette of colors obtainable from the printer and a pseudo-color image of data are presented in Figures 6 and 7 respectively. ACTPLT has been written in MACRO-11 specifically for RT-11. This software is heavily laden with RT-11 system programmed requests that perform a variety of the system specific functions. ACTPLT has been written in a format resembling the operation of an RT-11 standard utility. When ACTPLT is started, the adequacy of the system resources are checked. First the LP handler is pulled into memory with a .FETCH and associated with a channel using a .ENTER programmed request. A failure of any of the above programmed requests or conditions will force a fatal error abort. Next, tests are performed to see if a frame buffer interface is installed on the system. An error here will just be flagged and shown as a warning so that the user can take proper action if plotting directly from the frame buffer is planned. This warning is not fatal since image files may still be

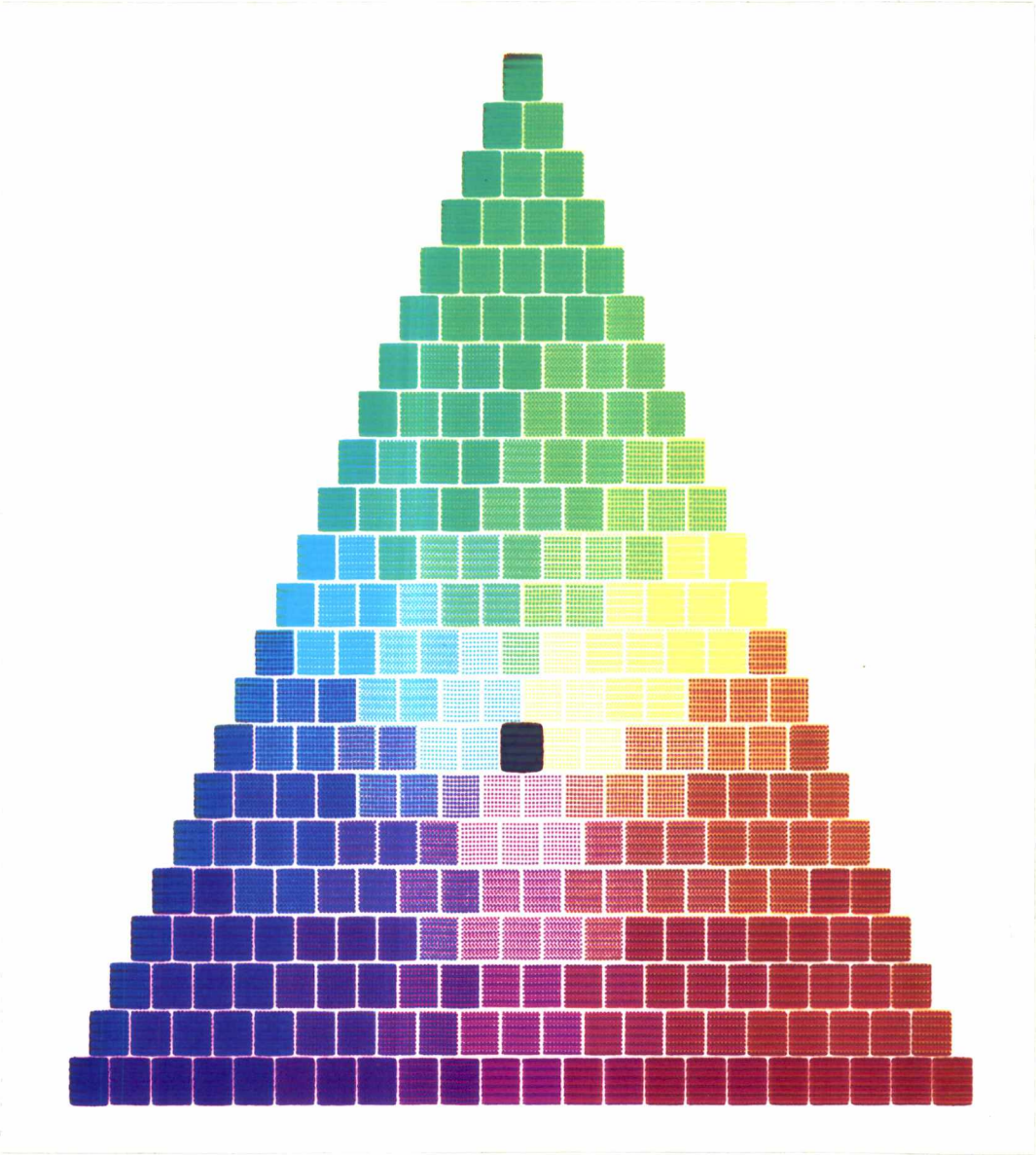


Figure 6. ACT II ink-jet plotter palette.

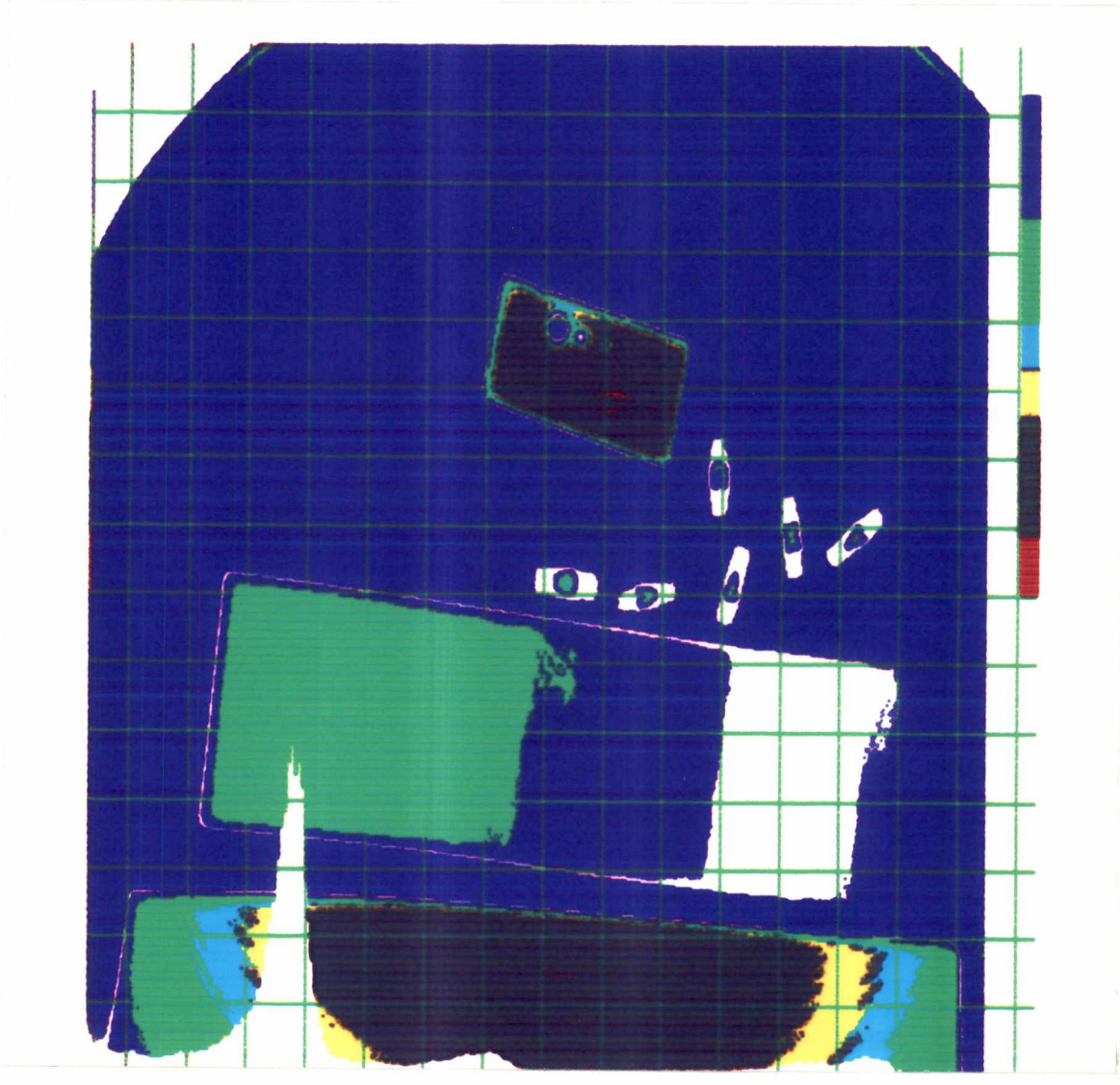


Figure 7. Ink-jet plotter pseudo-color image.

plotted. Now that the system resources have been evaluated, the program is ready for accepting the standard RT-11 utility command string input. After the command string has been properly input, the options list is converted to an option word for each item in the command string. Each item in the command string is now processed separately. The input command string is parsed for output and a check is made for bad options. If the item is to be plotted from a file, the file is checked for its existence. If the item is to be plotted from the frame buffer the initialization flag is checked. If a file or the frame buffer is present, the raster size code and default printing options are sent to the plotter, the option word is converted to an ASCII string and sent to the plotter, and the image is then plotted. Detailed ACTPLT user's information, including a program flow chart, is available in Appendix F.

CHAPTER 4

NOISE REDUCTION

The image quality of X-ray images can be greatly improved through noise reduction. Inherent to digitally acquired X-ray images is the fluctuation of brightness due to the randomness of the production and absorption of X-rays [6]. This phenomena, Quantum Mottle, must be considered in the X-ray imaging process. Image quality is also effected by the electronic and optical noise sources present in the imaging system [3]. Noise can be reduced and/or filtered using image processing techniques.

Frame Averaging

Because the scenes that are being digitized are stationary, multiple frames can be digitized and averaged to reduce the noise. Frame averaging, an effective noise suppressant, forces the random fluctuations of a single frame to cancel out relative to the steady-state signal arising from the actual X-ray shadow [10]. Frame averaging improves the Signal-to-Noise ratio by $\sqrt{N}$, where N repetitive frames are accumulated. This factor has been derived for signal-independent, additive, statistically stationary noise [10]. Frame averaging not only allows noise to be reduced in an image but it also increases the resolution and contrast. This improvement, however, is obtained only at the expense of a reduction in temporal resolution. Frame averaging is implemented very easily on the image processor. Powers of 2 digitized video frames are successively summed into the accumulation frame buffer.

The power of 2 constraint enables the averaging process to be completed by using the image processor's ALU barrel shifter to perform the division. With the assistance of the intensity profiles across the respective non-averaged and averaged images in Figure 8, the improvement of image quality can be ascertained.

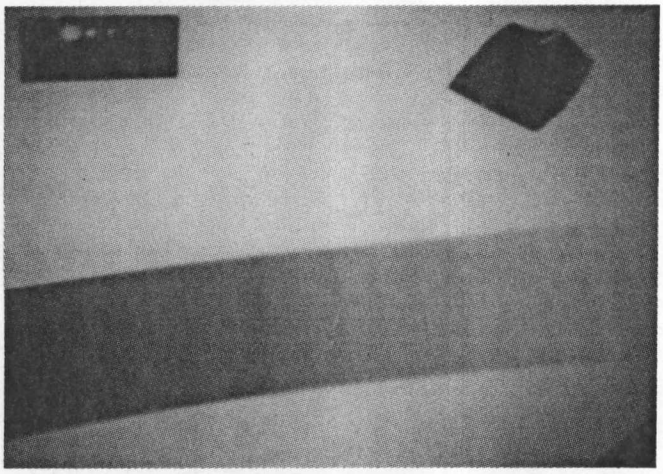
Adaptive Noise smoothing filter

Even though frame averaging is used effectively to acquire the reduced noise image in Figure 8, it can be seen that the image obtained by averaging 128 frames still contains a visible level of noise. Convolution with a low pass spatial filter can been performed to obtain a more smooth image. This method, however, reduces an already resolution deficient image by smearing edges and detail. To obtain reliable measurements, it is desirable to use a noise filtering technique that will not disturb the features of the images. Non-linear filtering allows noise to be filtered while preserving edge sharpness [11,12,13]. The filtered image

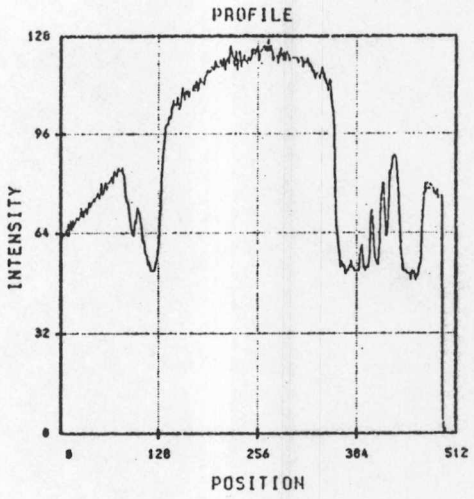
$$I_f(x,y) = (1-K(x,y))M(x,y) + K(x,y)I(x,y) \qquad (1)$$

is a weighted combination of the local mean representation of the image and the original image itself. The weighting factor is given as

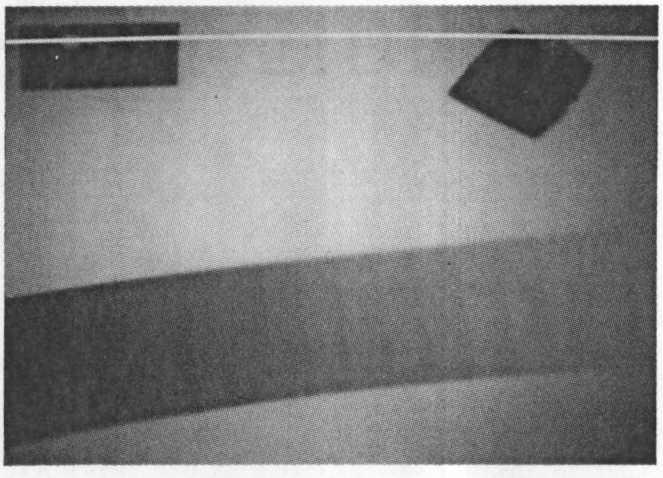
$$K(x,y) = V(x,y)/(V(x,y)+\sigma^2), \qquad (2)$$



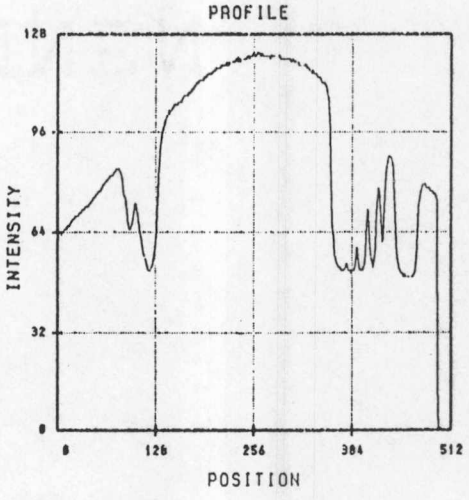
(a)



(b)



(c)



(d)

Figure 8. Effects of frame averaging on image quality. (a) One frame digitized, (b) Intensity profile of (a), (c) 128 frames averaged, (d) Intensity profile of (c).

where V is the local variance of the original image and σ^2 is its noise variance. The local mean used in equation (1) and the local variance used in equation (2) are given as

$$M(x,y) = \frac{1}{(2r+1)(2s+1)} \sum_{p=x-r}^{x+r} \sum_{q=y-s}^{y+s} I(p,q), \qquad (3)$$

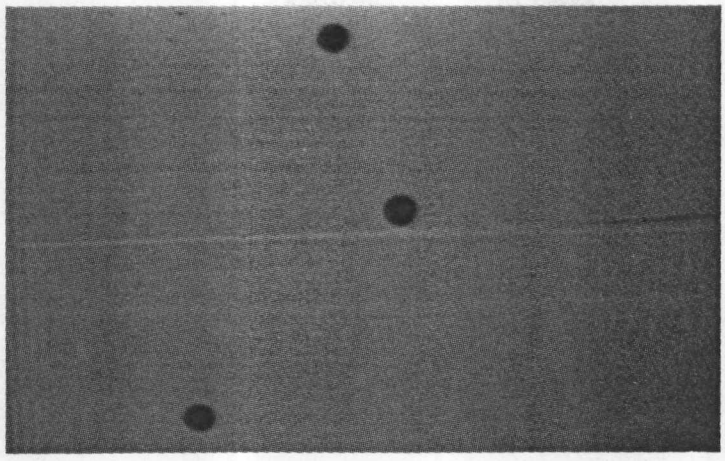
and

$$V(x,y) = \frac{1}{(2r+1)(2s+1)} \sum_{p=x-r}^{x+r} \sum_{q=y-s}^{y+s} (I(p,q)-M(x,y))^2 \qquad (4)$$

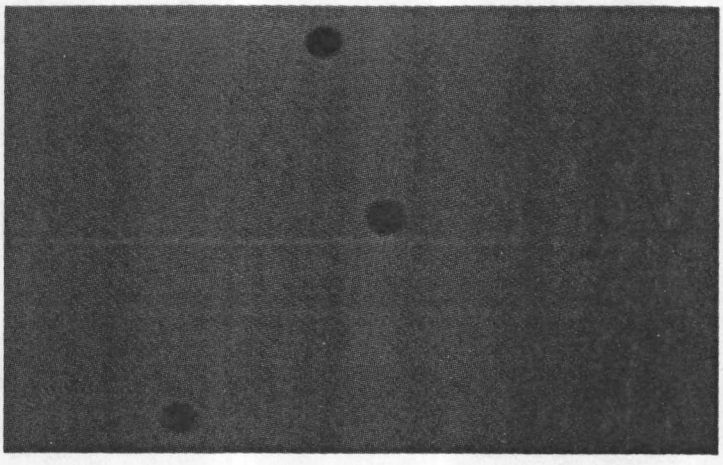
respectively. Intuitively, the reconstructed image of equation (1) becomes the original image at pixels where the local variance is much larger than the noise variance and becomes the smoothed image at pixels where the variance is much smaller than the noise variance. An example of the intermediate images generated by the adaptive noise smoothing filter is given in Figure 9. The local variance image should be particularly noted. The local mean and variance in equations (3) and (4) have also been used for content-dependent contrast enhancement of local features in X-ray images [14].

Adaptive Noise Smoothing Filter Implementation

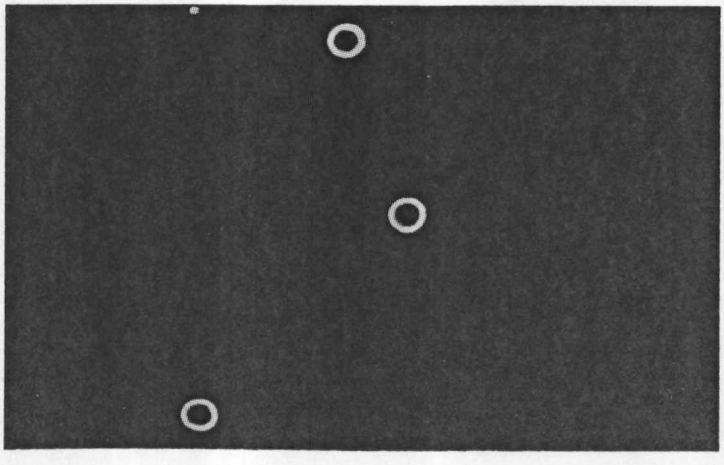
Obtaining the reconstructed image from the original image using the existing hardware on the system is best implemented using both the array processor and the image processor. Since the architecture implemented



(a)



(b)



(c)

Figure 9. Images of adaptive noise smoothing filter. (a) Original, (b) Local mean, (c) Local variance.

on the image processor allows several operations on a frame of data in a single pass, the local mean and local variance of an image and their combination can be performed on the image processor. The weighting factors will be applied by the array processor. The local mean is essentially the digital convolution of the image with the kernel values equal to 1. For digital images, convolution becomes a double sum. Digital convolution of an image is accomplished by moving the image buffer over the result buffer. Figure 10 demonstrates the neighborhood summing used to obtain the local mean of an image on the image processor graphically. The image buffer is initially misaligned in both the X and Y directions with respect to the resultant frame buffer according to the size of the neighborhood by adjusting the values in the image frame buffer pan and scroll registers. The image data is then summed to the data in the accumulation buffer in a single ALU pass. The accumulation result is obtained by successive horizontal and vertical slides of the image frame until the neighborhood has been covered. A 16 bit deep frame buffer has been configured as the accumulation buffer. This was established to prevent overflow occurring during the accumulation operation. Calculation of the local variance is obtained similarly by moving the image frame with respect to both the local mean image and the resultant frame. Calculation of the local variance is shown graphically in Figure 11. Division by the neighborhood size for both of the previously mentioned operations can be accomplished using the image processor. By applying a series of multiplications and shifts, the division is approximated.

$$M'(x,y) = \sum_{p=x-r}^{x+r} \sum_{q=y-s}^{y+s} I(p,q)$$

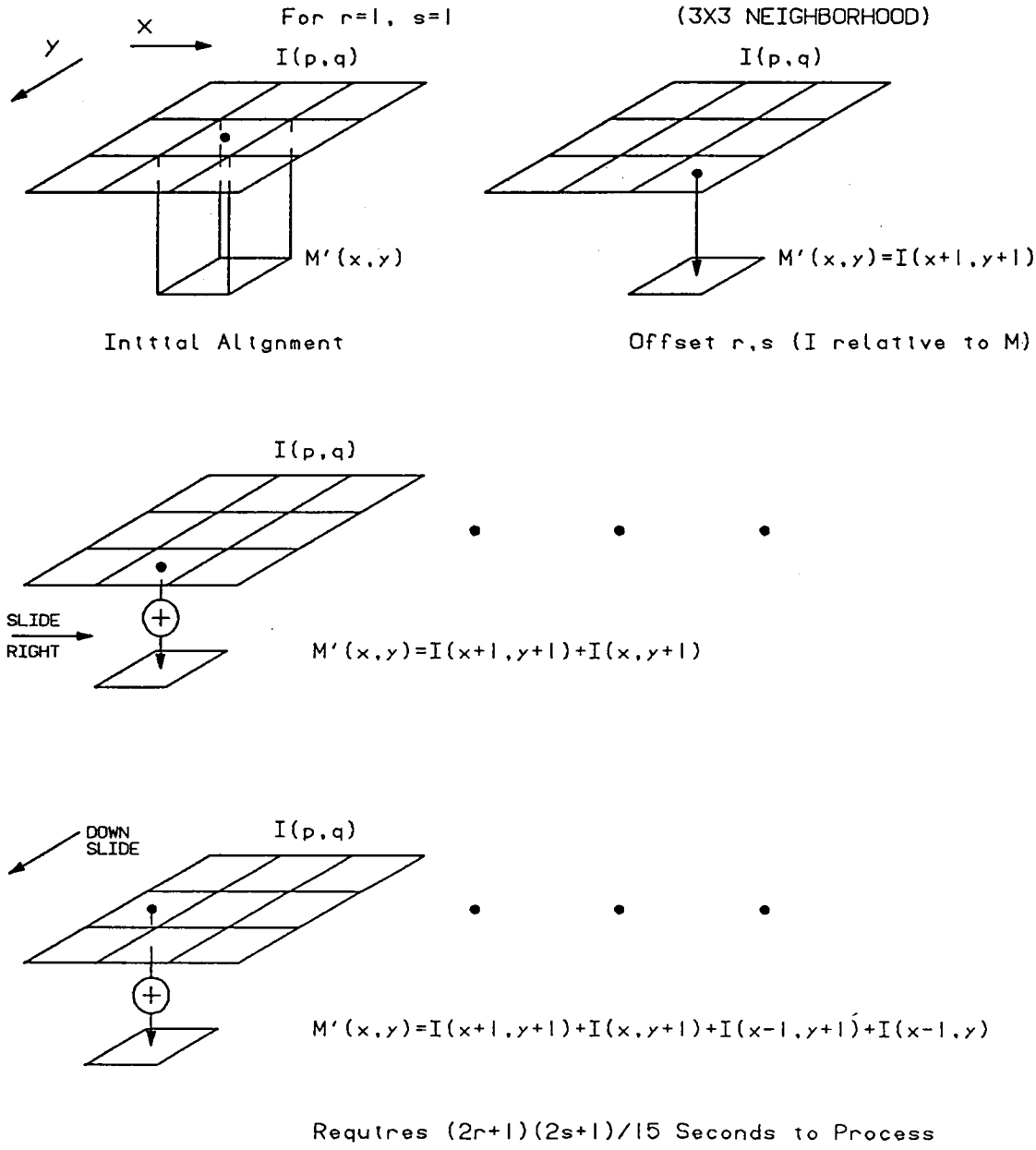


Figure 10. Convolution implementation on an image processor.

$$V'(x,y) = \sum_{p=x-r}^{x+r} \sum_{q=y-s}^{y+s} (I(p,q)-M(x,y))^2$$

$$= \sum_{p=x-r}^{x+r} \sum_{q=y-s}^{y+s} \underset{①}{I^2(p,q)} - \underset{②}{2I(p,q)M(x,y)} + \underset{③}{M^2(x,y)}$$

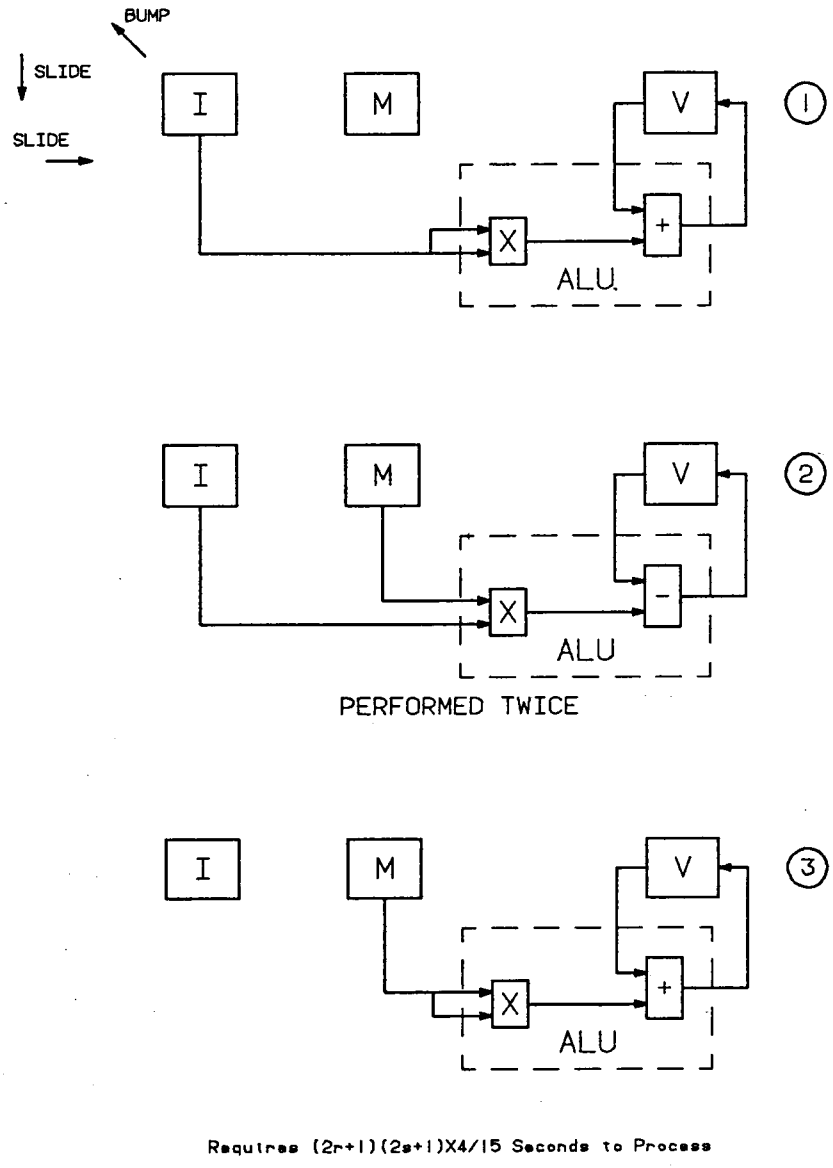
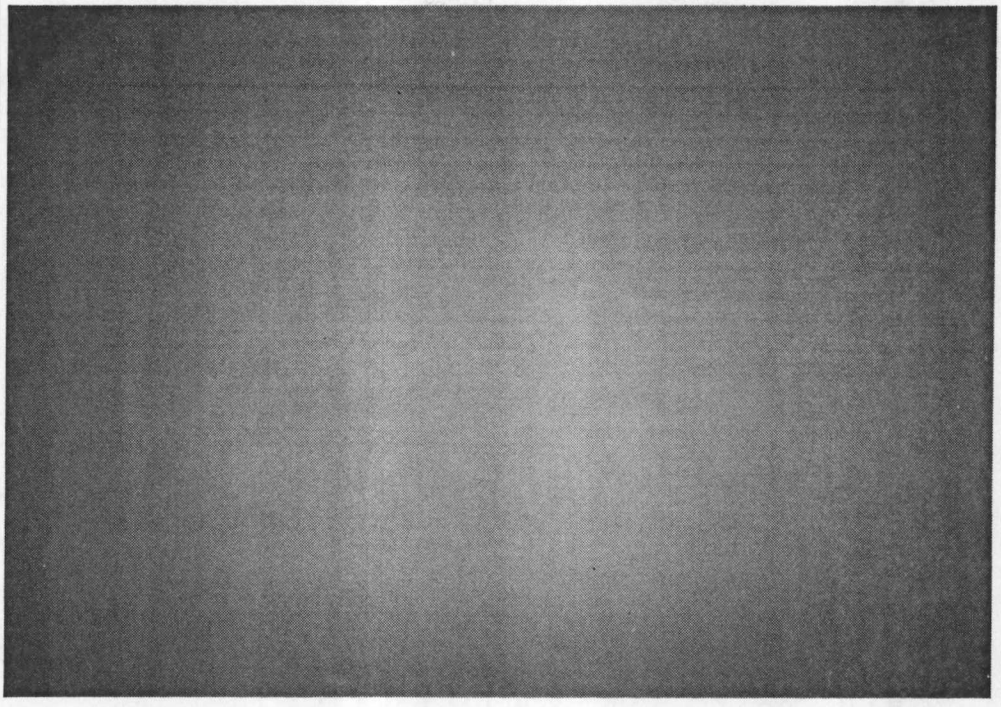


Figure 11. Local variance implementation on an image processor.

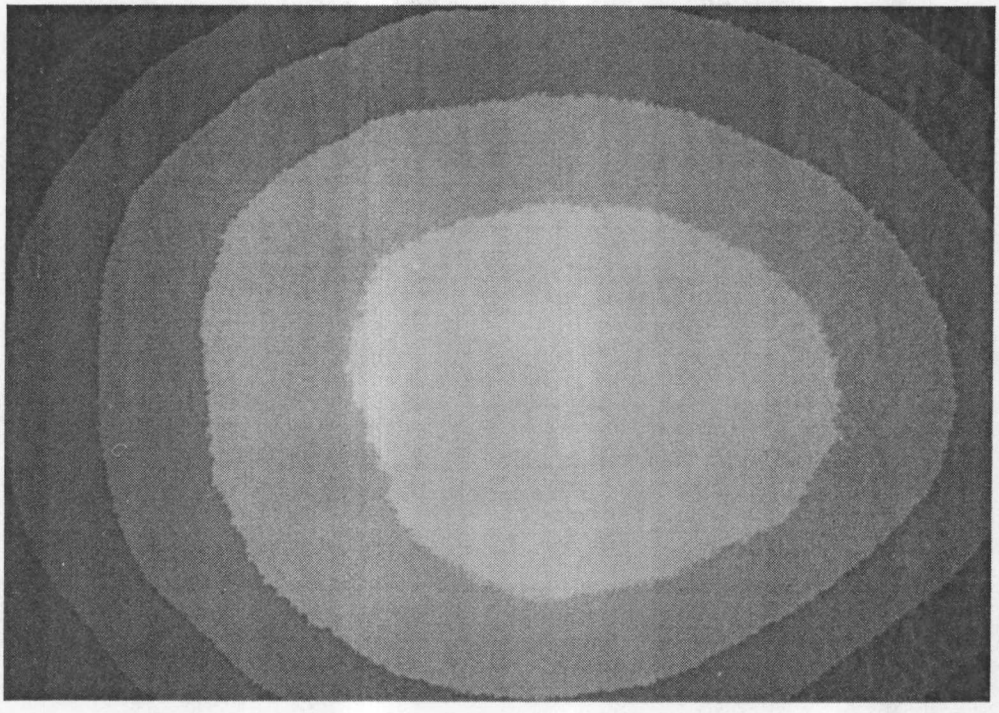
CHAPTER 5

FIELD-FLATTENING

It has been stated that field-flattening is an "absolutely crucial phase" [15] of automated analysis. The variation of thickness of the object under test, non-uniformity of the x-ray beam, or non-linearities of the detector require field-flattening to be performed. The non-uniformity of the detected X-ray intensity due to uneven X-ray illumination and detector non-linearities [16] will be dealt with here. The two images in Figure 12 were created to emphasize the non-uniform characteristics of the X-ray illumination. These images were acquired without any test object. The video detector was in direct alignment with the X-ray beam. When a global threshold is applied to the unevenly illuminated image in Figure 13 for scene segmentation, the gradients due to the uneven illumination cause the intensity level of the center test object in the image to fall below the selected threshold value. The grey levels of the gradient are comparitively significant to the grey levels of the test objects. In the field-flattened image in Figure 13, the center object is detected as easily as the outer test objects. Other than background subtraction, low frequency suppression in the frequency domain [17] and line-by-line polynomial-fit-subtraction in the spatial domain [15] are two methods that will accomplish adequate removal of the slowly varying background. The polynomial-fit-subtraction technique can be improved by using a test of significance of departure and reiteration of the fitting process until the fit is less influenced by objects in the image.

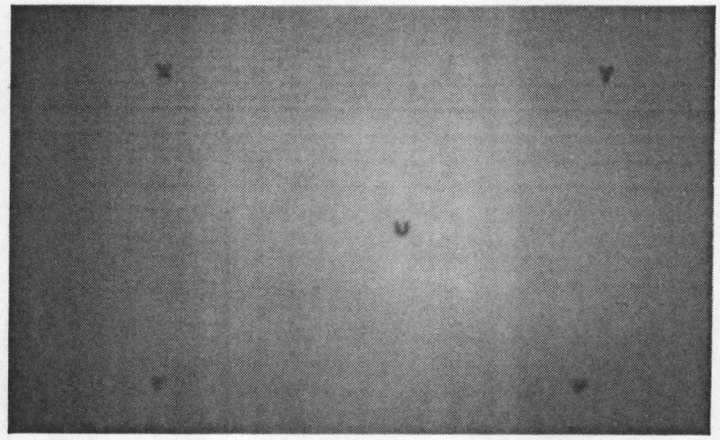


(a)

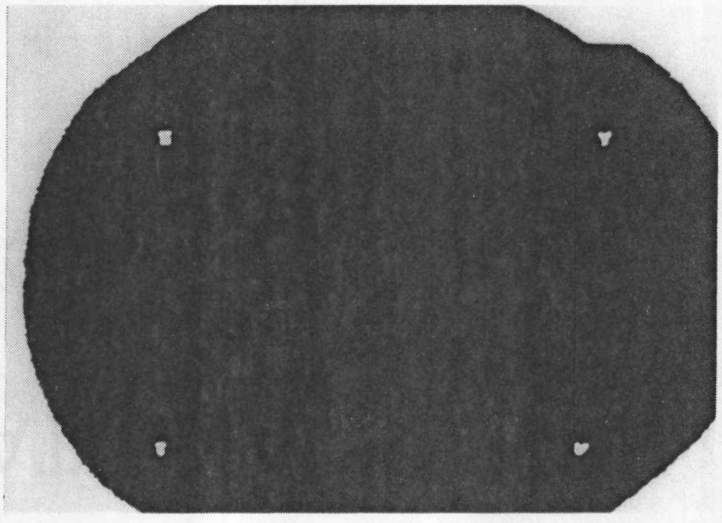


(b)

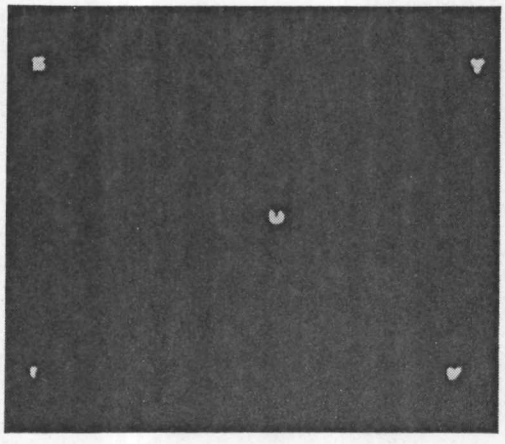
Figure 12. "Bull's eye" effect. (a) Original of X-ray intensity distribution, (b) Contour of (a).



(a)



(b)



(c)

Figure 13. Threshold applied to a non-field flattened image. (a) Original, (b) Original tresholded (c) Field-flattened, thresholded image.

Background Subtraction

Background subtraction permits a stored reference image to be subtracted from another image to obtain only the information of interest in the difference image. Performing background subtraction from the working images was the first approach taken toward obtaining a flat-field. To relate the actual change in material thickness of the defects to the changes in the detected X-ray intensities, a simple subtraction can not be performed. Referring back to Figure 3, the intensity distribution of the X-rays detected at the image plane are related to the thickness of the material

$$I(x,y) = I_o(x,y)\, e^{-\mu\rho Z(x,y)} \qquad (1)$$

where I_o is the intensity distribution of the illumination of the X-ray source at the object plane, Z is the distribution of the material thickness, ρ is the material density, and μ is the atomic absorption coefficient dependent on the X-ray energy and material composition. Let the background intensity distribution and the working image distribution be

$$I_b(x,y) = I_o(x,y)\, e^{-\mu\rho Z_b(x,y)}$$

and (2)

$$I_w(x,y) = I_o(x,y)\, e^{-\mu\rho Z_w(x,y)}$$

respectively. The effective thickness information can be obtained by

taking the ratios of equation (2)

$$\frac{I_b(x,y)}{I_w(x,y)} = \frac{e^{-\mu\rho Z_b(x,y)}}{e^{-\mu\rho Z_w(x,y)}} \quad . \tag{3}$$

Taking the natural log of both sides of equation (3) leaves

$$\text{Ln}\,(I_b(x,y)) - \text{Ln}\,(I_w(x,y)) = \mu\rho\{Z_w(x,y) - Z_b(x,y)\} \tag{4}$$

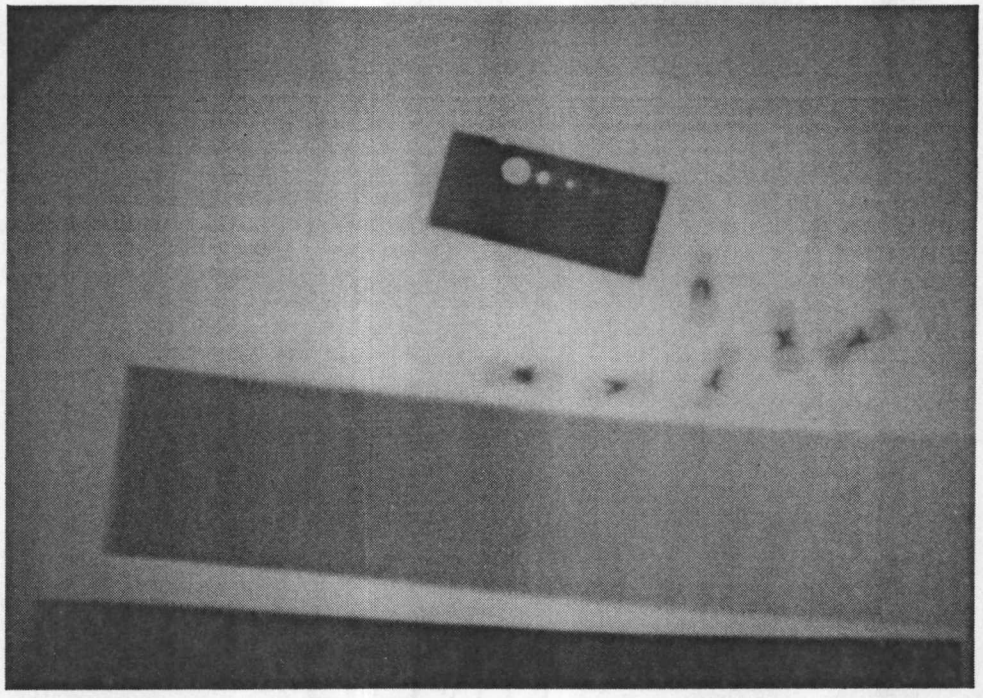
The effective thickness image is thus

$$\Delta Z(x,y) = \frac{1}{\mu\rho}\ \{\text{Ln}\,(I_b(x,y)) - \text{Ln}\,(I_w(x,y))\}. \tag{5}$$

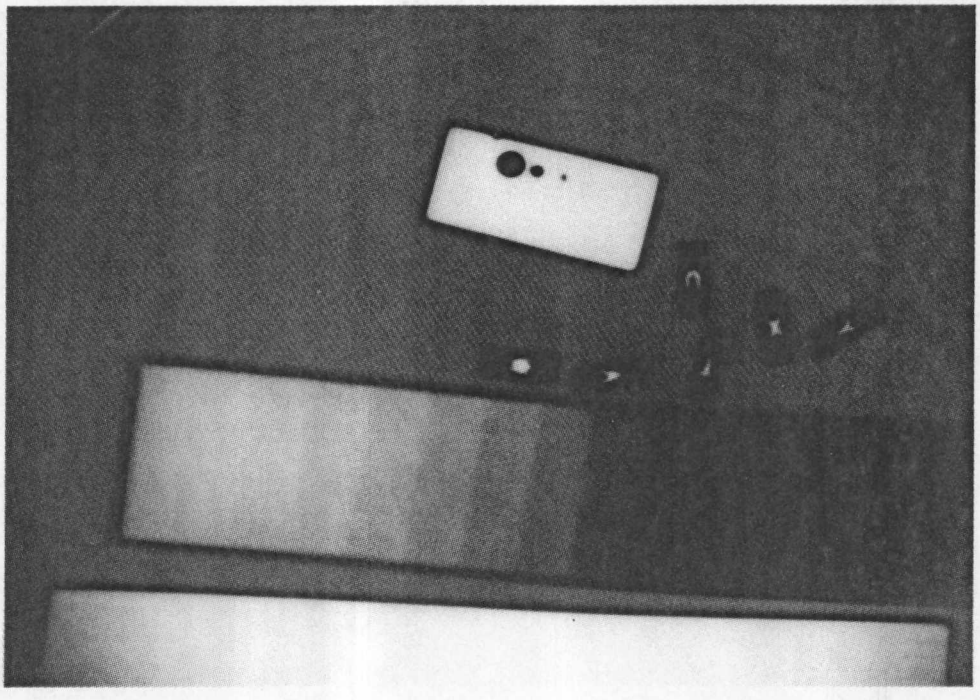
The effectiveness of background subtraction is demonstrated on the test image in Figure 14.

Polynomial-Fit-Subtraction

Variations of the x-ray illumination with time, either due to X-ray tube warm-up or operator adjustment, have caused background subtraction to perform inadequately for a given stored reference image. A technique was needed that could create an approximate background reference image from the working image. The polynomial-fit-subtraction technique has been used in the automation of defect detection in radiographs of the high-explosive charge region of artillery shells [18]. This technique uses a polynomial fit of the working image to approximate the gradients



(a)



(b)

Figure 14. Background subtraction. (a) Initial image, (b) Difference image obtained by logarithmic subtraction of (a) and Figure 12(a).

in the image. This approximation is then subtracted from the working image to flatten the field and improve the ability to detect the defects. The applicability of this technique is largely dependent on image content. Large items in the image will effect the fit and will be removed when the subtraction is performed [15]. Steep gradients coupled with an inadequate polynomial to fit them will cause large residuals to be misinterpretted [15]. A higher order polynomial could be implemented for the case of steep gradients that would effectively reduce the false residuals but would also include the objects. Improving the ability to detect small defects in slabs of material allow this technique to be a viable candidate. The test object has no features and is of uniform thickness. To adapt this technique for this work, the X-ray intensity distribution is approximated with a polynomial as if no defects are present in the image. This fitted intensity becomes the background reference image for the operation required in equation (5). If an image containing n+1 pixels in a horizontal line is assumed, the intensity value of pixel x on line y can be approximated with the mth-degree polynomial, m<n,

$$P_y(x)=A_m x^m + A_{m-1}x^{m-1} + \cdots + A_2 x^2 + A_1 x + A_0. \qquad (6)$$

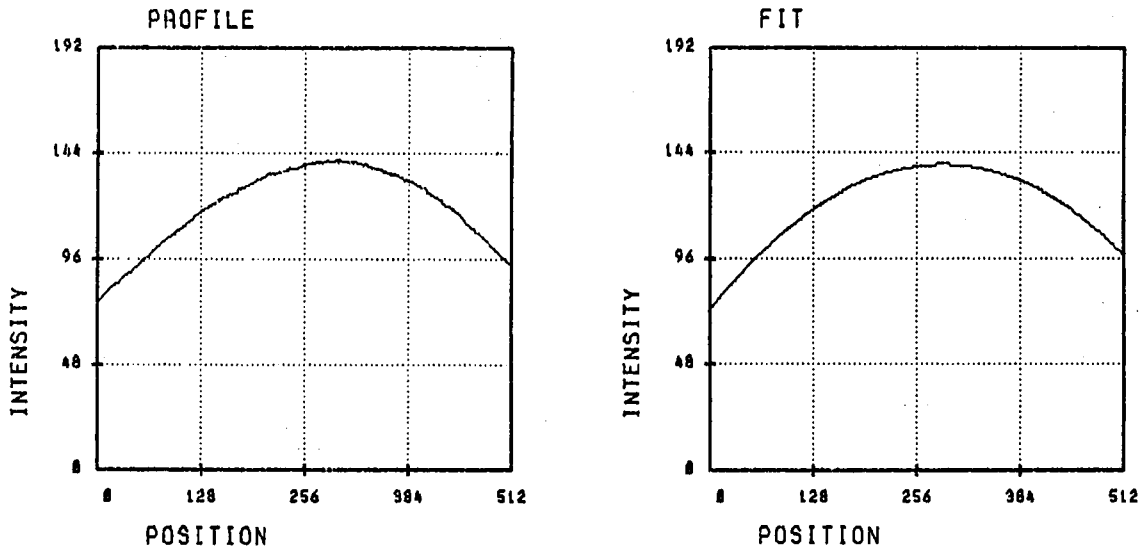
From experimentation, the X-ray illumination variation is adequately represented by the 2nd-order polynomial.

$$P_y(x) = A_2 x^2 + A_1 x + A_0. \qquad (7)$$

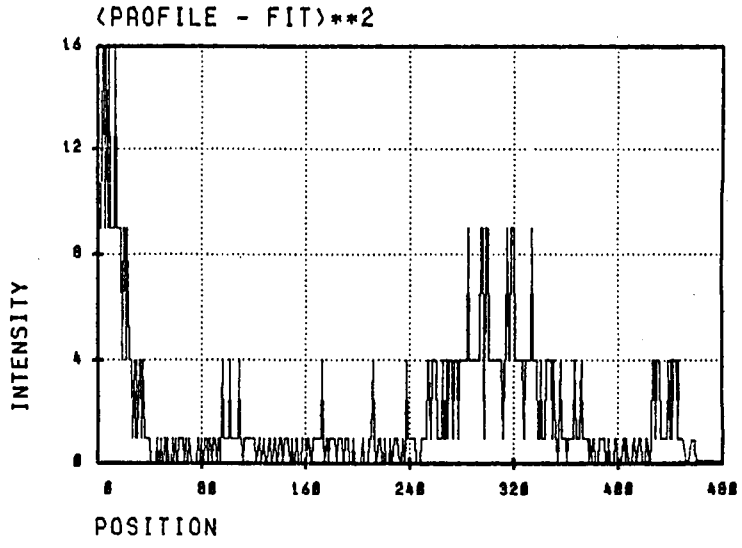
An intensity profile taken across an image of the X-ray distribution can be compared to the 2nd-order fit in Figure 15. The difference squared between the two profiles has also been plotted. The low order of the polynomial should insure that the expected defects are not included in the fit.

Two variations to the polynomial-fit-subtraction technique have been devised that provide visually discernable improvements. One involves a decision based algorithm that eliminates points that deviate greatly from the fit and are subsequently indicative of features. Since the least squares curve fit is based on the sum of the squares of the departures being a minimum, a measure of departure founded on this numerical value can be used to eliminate points in the profile that should be ignored when the polynomial fit is reperformed. This value gives a significance of how well the fit represents the intensity profile. If this value is large it is likely that the intensity line crosses a defect. In this case a threshold value can be established from apriori knowledge from a training period of the statistics that allows points of the defect to be eliminated before a second fit is attempted. The other variation to this technique involves grossly smoothing the image prior to performing the fit.

To demonstrate polynomial-fit-subtraction, the image in Figure 16 will be used. The steps of the process for a particular profile line across a penetrameter in the image are given in Figure 17. Figure 18 shows the steps used to obtain a more field flattened line when the significance of departure is utilized. The histogram of the original image and the histograms of the two field-flattened images are given in

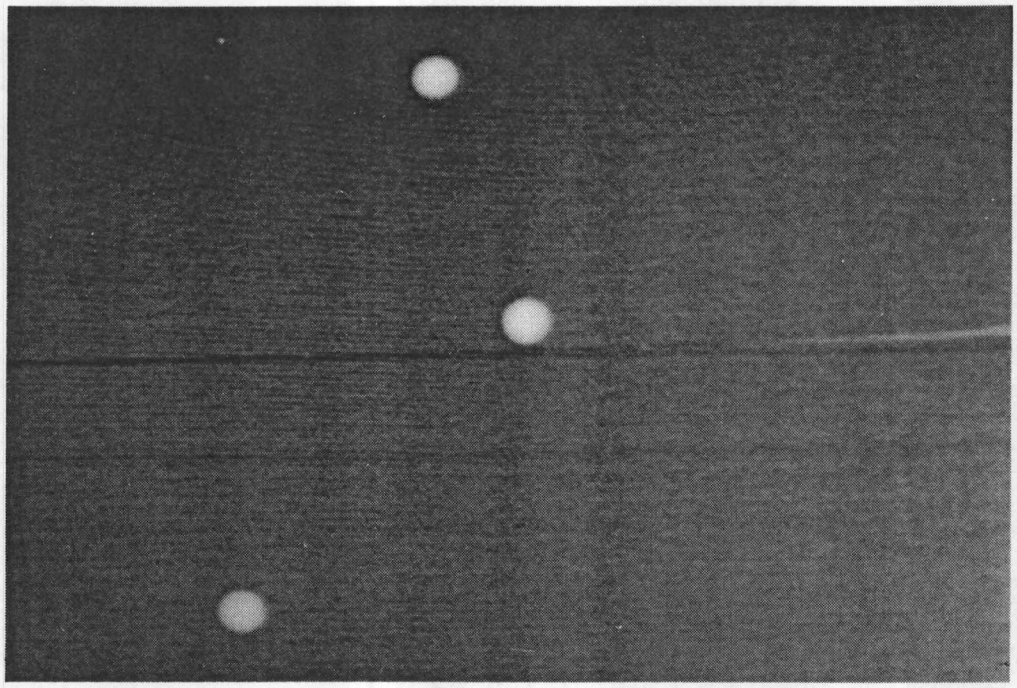


(a) (b)

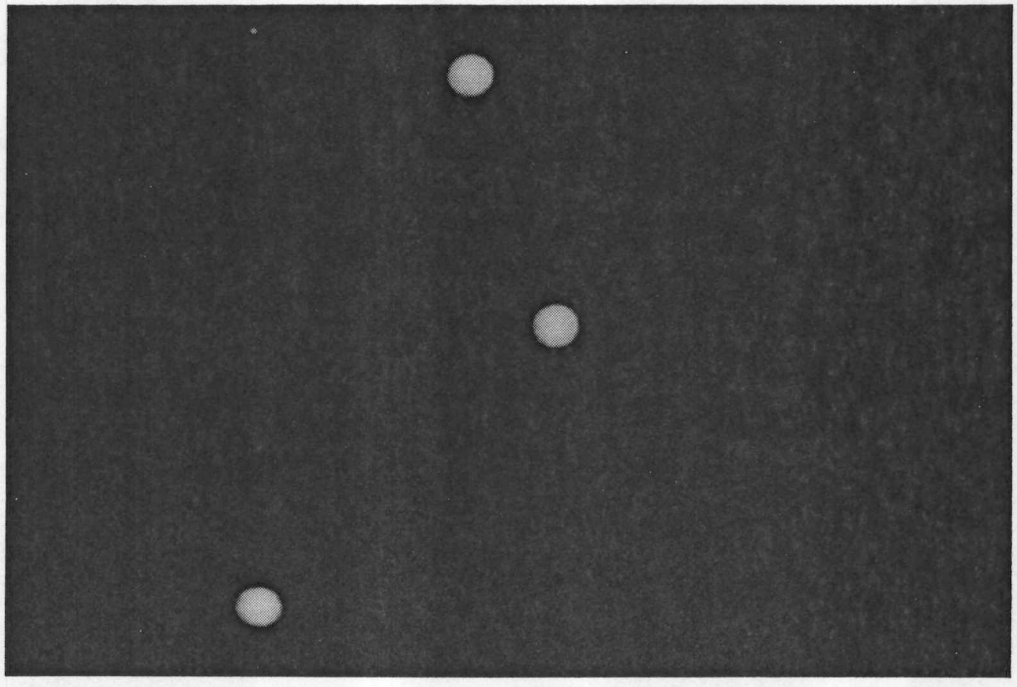


(c)

Figure 15. Polynomial fit to X-ray distribution. (a) Intensity profile of X-ray distribution, (b) 2nd-order polynomial fit to (a), (c) Squared error difference between (a) and (b).

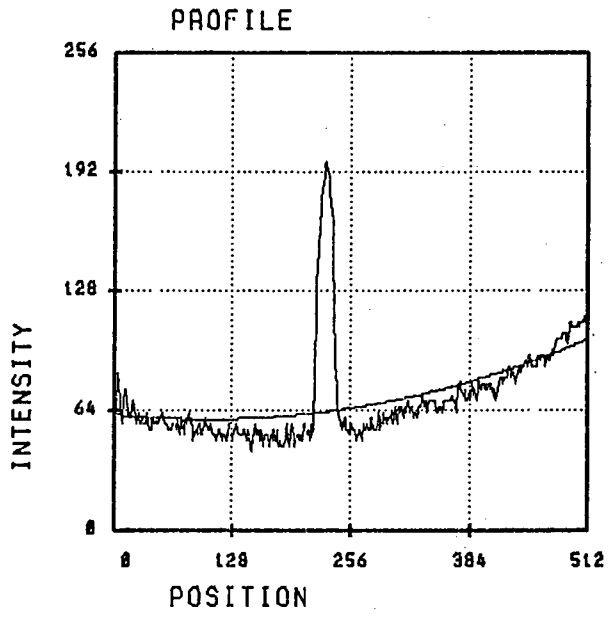


(a)

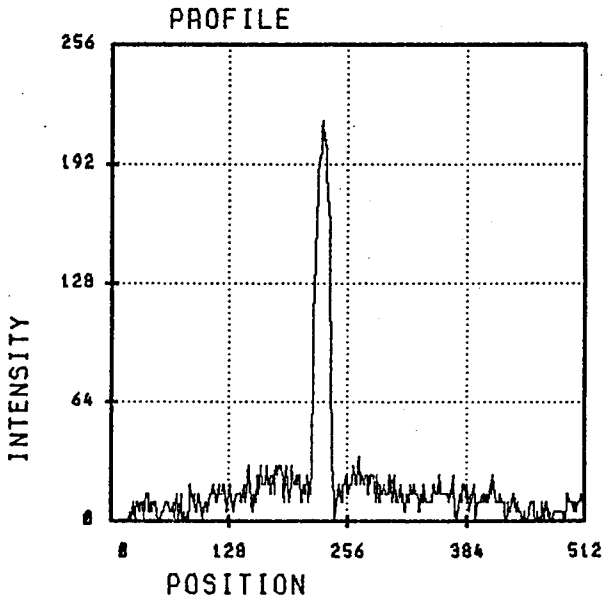


(b)

Figure 16. Field-flattened 20-mil penetrameter image. (a) Original, (b) Field-flattened.



(a)



(b)

Figure 17. Example curve fit intensity profiles. (a) Original intensity profile with fit, (b) Residues from fit subtracted from original intensity profile.

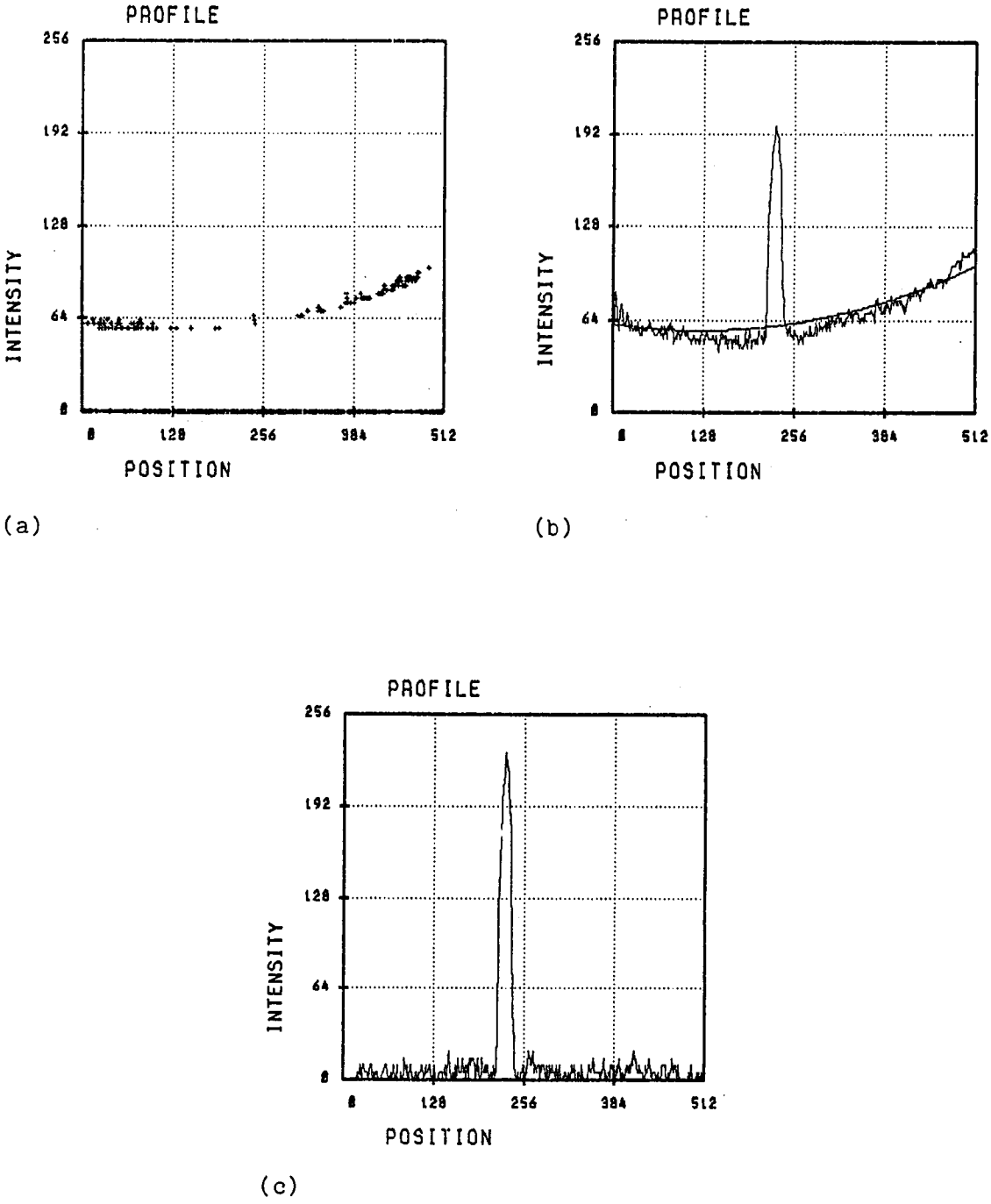


Figure 18. Profiles using significance of departure. (a) Points eliminated from intensity profile, (b) Original intensity profile and fit, (c) Residues.

Figure 19. The histogram of the original image is essentially unimodal (characteristic of images lacking in visible detail), while those that have been field-flattened have a more pronounced tail indicating the intensity distribution of the flaws. Some measures of First-order probabilistic differences in image tone have been computed from the image histograms and are contained in Table 3. From the histogram function

$$H(i),\ i = 0,\ n-1, \tag{8}$$

where n possible gray levels are allowed, the statistical values used in Table 3 are determined from the r-th raw moment and central moment

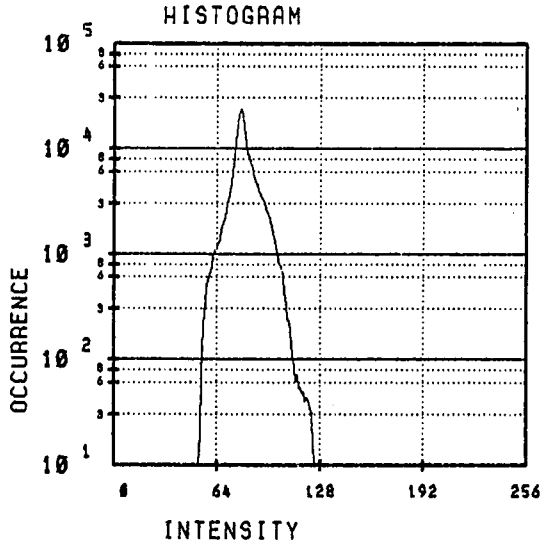
$$M_r = \sum_{i=0}^{n-1} i^r\, H(i) \tag{9}$$

and

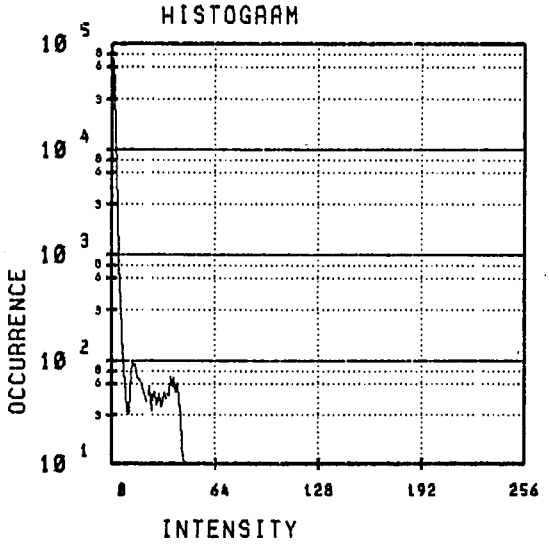
$$C_r = \sum_{i=0}^{n-1} (i-M_1)^r\, H(i)\ . \tag{10}$$

The variance, Skew, and Excess are given as

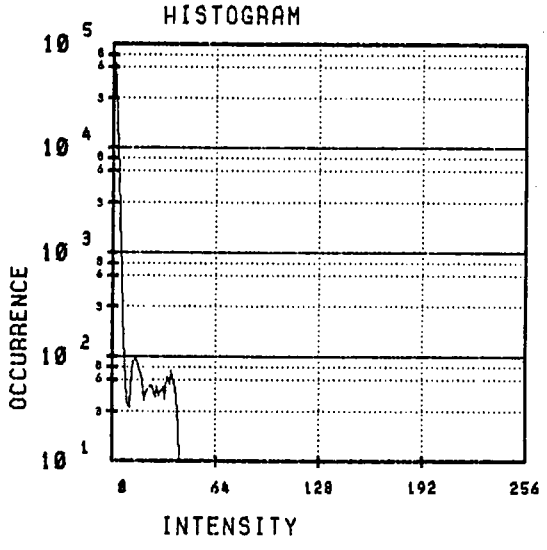
$$C_2 = M_2 - M_1^2\ , \tag{11}$$



(a)



(b)



(c)

Figure 19. Histogram comparisons. (a) Original histogram of image in Figure 16 (a), (b) Histogram of field-flattened image in Figure 16 (b), (c) Histogram when significance of departure is used.

Table 3

Effects of Field-Flattening
On Histogram Statistics

Image	Mean	Variance	Skew	Excess
Original Image	81.2	68.9	0.58	4.7
Single pass fitted	1.2	6.25	9.6	120.
Two pass fitted	1.1	6.25	11.0	140.

$$S = \frac{C_3}{(C_2)^{3/2}} , \tag{12}$$

and

$$E = \frac{C_4}{(C_2)^2} . \tag{13}$$

From Hall, the variance is a measure of dispersion and is related to image contrast, the skewness is a measure of asymmetry about the mean, and the coefficient of excess is a measure of the distribution's tendency to cluster about its mean or disperse towards its extremes.

Polynomial-Fit-Subtraction Implementation

Computational performance for the polynomial-fit-subtraction technique is obtained by "precomputing" [15] the least squares fit values. Performance for the iterative fitting algorithm is obtained by subtracting the component of the eliminated points from the precomputed values. From a Least Squares Curve fit, there will be three equations to solve for the three unknown coefficients in equation (7). In augmented matrix notation, these equations are

$$\begin{bmatrix} A_{02} & A_{01} & A_{00} & \Sigma\, I(x) \\ A_{12} & A_{11} & A_{10} & \Sigma\, I(x)x \\ A_{22} & A_{21} & A_{20} & \Sigma\, I(x)x^2 \end{bmatrix}. \tag{14}$$

where I(x) is the intensity at the pixel coordinate x in the line being fitted. Gaussian elimination for simultaneous equations is then used to manipulate equation (14) to obtain

$$\begin{bmatrix} A_{02} & A_{01} & A_{00} & \Sigma\, I(x) \\ 0 & A'_{11} & A'_{10} & \Sigma\, I(x)x - b_1\, \Sigma\, I(x) \\ 0 & 0 & A''_{20} & \Sigma\, I(x)x^2 - b_3\, \Sigma\, I(x)x \\ & & & \quad + (b_3 b_1 - b_2)\, \Sigma\, I(x) \end{bmatrix}. \qquad (15)$$

All of the values in equation (15) except $\Sigma\, I(x)$, $\Sigma\, I(x)x$, and $\Sigma\, I(x)x^2$ are specified for a given line length and are the same for each line in an image. After these values are calculated, the coefficients can be determined and the polynomial can be evaluated to obtain the approximation to the background.

Use of the array processor is critical to improve the speed in obtaining the sums and evaluating the polynomial. Not only just using the array processor speeds up execution, but the manner in which it is used also can improve execution performance. Normally the array processor routines operate on an array of data and return that data to a user specified buffer for each subroutine call. Direct manipulation of the array processor, sending it operation codes and source and destination addresses, allows intermediate results to be maintained in the array processor's internal accumulator before a final result is obtained. Using the accumulator eliminates the need of writing an

intermediate result into a temporary buffer and reading it back into the array processor for the next operation. Another performance improvement is attained by letting the array processor read data directly from the image frame buffer's data register. This mode of operation is possible only due to the hardware features afforded by both the array processor and the image processor's frame buffers. Pixel data access from a frame buffer is typically accomplished through the data register in the I/O page after presetting the X and Y address registers for the desired pixel location. Once the starting location of a pixel is established, the frame buffers auto-increment mode enables successive pixel access just by reading the pixel data register. The array processor has been configured to allow its source and destination addresses to map to devices in the computer I/O page. A configurable jumper physically located on the array processor forces the Q-bus address line BBS7L to be asserted when the appropriate bit is set in the source/destination address register. This Q-bus bus address line, Bank 7 select, indicates to devices on the bus that the address present on the bus is to be decoded by devices in the I/O page. With out this feature, data in frame buffers would have to be moved to a temporary storage location before the array processor could access it. Table 4 shows the execution time improvements afforded by the previously discussed techniques. Poly computational time could be further reduced by using fewer points per line and/or by using less lines for the fit. The instructions used to manipulate the array processor directly to achieve the results needed for the polynomial-fit-subtraction are given in Figure 20.

Table 4

Evolution of Improvements In
Polynomial-Fit-Subtraction Execution Speed

Method	Execution time In Seconds
Implemented in Fortran	125
With Normal Array Processor Execution	21
With Direct Array Processor Access	16
With DMA To/from Frame Buffer	12
With Subtraction by Image Processor	10

Calculation of $\Sigma I(x)$, $\Sigma I(x)x$, and $\Sigma I(x)x^2$.

Array Processor OP-Code	OP-Code Description	Action
014	Convert Fraction To Floating	Frame buffer data to Accumulator (ACC)
034	Sum and store Elements	SUM = Σ ACC(X)
222	Multiply and Sum products	SUM = Σ ACC(X)X
222	Multiply and Sum Products	SUM = Σ ACC(X)X^2
To evaluate polynomial		
110	Scalar Vector Multiply	TEMP(X)=XA_1
114	Scalar Vector Multiply	ACC(X) = X^2A_2
064	Add Accumulator To Vector	ACC(X) = ACC(X) + TEMP ACC(X) = $A_2X^2 + A_1X$
074	Add Accumulator To Scalar	ACC(X) = ACC(X) + A_0 ACC(X) = $A_2X^2 + A_1X + A_0$
024	Convert Floating Point to Fraction	Move accumulator data to Frame buffer

Figure 20. Implementing Summations and Polynomial evaluation directly on the array processor.

CHAPTER 6

IMPURITY DETECTION

The culmination of this effort is the computer aided mensuration of the area of simulated impurities within sheets of material. The impurities have been simulated by placing 0.020" Aluminum spheres on a sheet of material being inspected. The effectiveness of the image processing techniques has been evaluated as to how well the area measurements compare to the known penetrameter size.

The image processing steps involved in inspecting the material are contained in Figure 21. The image is acquired using frame averaging. Averaging 128 frames requires 4.3 seconds yet improves the signal-to-noise ratio by a factor of 11. The preprocessing step is necessary to prepare the image for segmentation. This step is discussed in more detail later. The scene is then segmented into a binary image using a global threshold. The threshold selection is presently the only step in the inspection process requiring operator intervention. After histogram interpretation, the operator selects an initial threshold. This threshold value is then varied until reaching a value immediately above where noise is introduced. Methods of threshold selection described in Reference 19 are being investigated to alleviate all operator intervention except material loading. As the last step in the inspection process, the regions indicating impurities are located and measured and a status report is generated.

The measurement results of three Aluminum spheres and a true impurity, as effected by the preprocessing steps and combinations of

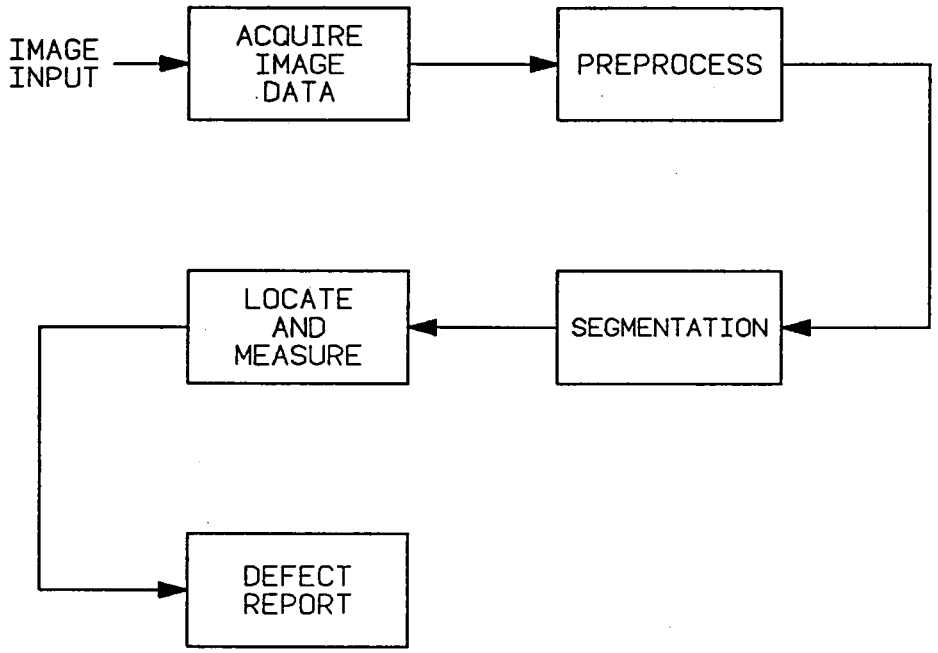


Figure 21. Automated inspection steps.

them, are given in Table 5. The test spheres should present an X-ray shadow of approximately 447 pixels. The gradients in the original image force a high threshold to be selected and thus obscure the impurities depending on where they are located in the image. By flattening the field using the polynomial-fit-subtraction, a global threshold has more intuitive sense and much better measurements are made. Through the iterative process of fitting the image, eliminating points, and refitting, the areas do approach the expected size but not enough to justify the added processing time required. When thresholding the field flattened image, it was evident that noise in the background was forcing a high threshold value to be used. To eliminate that noise, adaptive noise smoothing and neighborhood averaging were attempted. From the data in Table 5, it can be seen that neighborhood averaging greatly expanded the areas of the impurities. The adaptive noise smoothing filter reduced the noise level in the background so that a lower threshold could be established without the loss of resolution in the edge regions. The joint preprocessing step that has been selected on the basis of measurement results is the adaptive noise smoothing filter and polynomial-fit-subtraction.

TABLE 5

Penetrameter Area Measurement Results[a]
(No. Of Pixels)

Image	Global Threshold	Real Defect	Penetrameters 1	2	3
Original	197	**[b]	201	285	271
Poly[c]	11	8	419	421	431
Poly2[d]	11	8	427	436	443
Mean Poly[e]	9	13	466	459	478
ANSF Poly[f]	8	11	441	446	458

[a] A 0.020" sphere should have an area of 447 pixels.

[b] ** Denotes flaw not found.

[c] Polynomial-fit-subtraction

[d] Two pass polynomial-fit-subtraction

[e] 5X5 kernel neighborhood smoothed/Polynomial-fit-subtraction.

[f] 5X5 adaptive noise smooth filtered/Polynomial-fit-subtraction.

CHAPTER 7

CONCLUSIONS

A general introduction to image processing's usefulness in electronic radiography has been presented. Additionally, an electronic radiographic inspection system has been described and implemented to measure impurities within a material. System specific software was written to enable better use of the hardware's resources and features. Much of this software has already been implemented on other systems where large arrays of data are processed. Hardware components evaluated on this system have also been utilized in other systems. ERIS has been designed to be very flexible. Additional input/output interfaces may be added as needed for additional image analysis. Additional disk drives may be installed to provide extra room for the archival of image data and storage of image processing programs. The improvements to image quality by digital image processing are essential for computer analysis of images presented by electronic radiography.

LIST OF REFERENCES

LIST OF REFERENCES

1. W. B. Boone, F. Schlieper, and B. Kinchen, "A New Second Generation Real Time Imaging System," QualTest-2 Conference Proceedings, October 25-27, 1983.

2. M. H. Jacoby, "Automatic Evaluation of X-ray and Acoustic Images," ASNT Paper Summaries, March 22-25, 1982.

3. H. Roehrig, H.D. Fischer, M.M. Frost, S. Nudelman and M.P. Capp, "Photoelectronic Imaging for Radiology," IEEE Transaction on Nuclear Science.,vol No. 6-78, February, 1981.

4. D. R. King, "Modular Hardware Simplifies Design of Image Processors," Computer Technology Review, Vol VI, No. 1, 1985.

5. R. H. Bossi, "Real-Time Radiography," LLNL Technical Report, UCRL-53091, February, 1981.

6. C. G. Gardner, "Automated Radiography A State-of-the-Art Survey," Nondestructive Testing Information Analysis Center Technical Report, NTIAC-78-1, June, 1978.

7. P. F. Henderson and D. M. Ting, "A System Builders Approach to Image Processing," Electronic Imaging, November/December 1982.

8. M. A. King and P. W. Doherty, "Array Processors In Nuclear Medicine," Electronic Imaging, November/December 1982.

9. M. E. Lapides, "Techniques for Efficient Storage and Retrieval of Industrial Radiographs," Electric Power Research Institute Technical Report, EPRI NP-2383, May, 1982.

10. K. R. Castleman, Digital Image Processing, New Jersey: Prentice-Hall, 1979.

11. D. T. Kuan, A.A. Sawchuck, T.C. Strand, and P. Chavel, "Adaptive Noise Smoothing Filter For Images With Signal Independent Noise," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-7, No. 2, March, 1985.

12. J. Lee,"Digital Image Enhancement And Noise Filtering By Use of Local Statistics," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 2, March 1980.

13. J. Lee, "Refined Filtering of Image Noise Using Local Statistics," Computer Graphics and Image Processing, Vol. 15, 1981.

14. G. Gaibotto, "Digital Processing of X-ray Images," 1983 International Electrical Electronics Conference.

15. M. H. Jacoby, "Image Data Analysis," The Nondestructive Testing Handbook on Radiography and Testing, Columbus, Ohio: American Society for Nondestructive Testing, 1982.

16. B. K. Bucklen, "Digital Signal Processing Techniques Compensate for Video Anomalies," Electronic Imaging, November/December 1982.

17. B. R. Hunt, D. H. Janney, R. K. Zeigler, "Introduction to Restoration and Enhancement of Radiographic Images," Los Alamos Scientific Laboratory Technical Report, LA-4305, April, 1970.

18. J. J. Pearson, Eppler, Firschein, Jacoby, Keng, and, Jaffey, "Automatic Inspection of Artillery Shell Radiographs," Twenty-Second Annual International Technical Symposium, SPIE, San Diego (1978).

19. J. Kittler, J. Illingworth, and J. Foglein, "Threshold Selection Based on a Simple Image Statistic," Computer Vision, Graphics, and Image Processing, No. 30, 1985.

20. R. C. Gonzalez and P. Wintz, Digital Image Processing, Reading, Mass.:Addison-Wesley, 1977.

21. E. L. Hall, Computer Image Processing and Recognition, New York: Academic Press, 1979.

22. W. K. Pratt, Digital Image Processing, New York: Wiley and Sons, 1978.

APPENDIXES

APPENDIX A

COMPONENT COSTS (NO CAMERA/MOTION CONTROL/IMAGE HARDCOPY)

CPU	1,395.00	
Serial line with Boot	495.00	
4 Mbyte Memory	1,350.00	
Mass storage controller	1,495.00	
Array Processor	4,995.00	
	9,730.00	
4 Frame buffers @1995.00	7,980.00	
Analog Processor	2,995.00	
ALU	1,995.00	
Histogram/Feature	1,995.00	
	14,965.00	
73 Mbyte Disk	2,895.00	
Terminal	1,200.00	
Computer Enclosure	1,600.00	
Instrument Rack	695.00	
17" Monochrome Monitor	995.00	
	7,385.00	
		$ 32,080.00

APPENDIX B

SYSTEM ENCLOSURE POWER BUDGET

DEVICE	AMPS AT +5V		AMPS AT +-12V	
	USED	AVAIL-ABLE	USED	AVAIL-ABLE
		50.0		8.0
CPU	4.5	45.5	--	8.0
SERIAL LINE	2.5	43.0	.1	7.9
4 MB MEMORY	2.6	40.4	--	7.9
MASS STORE	6.4	34.0		7.9
ARRAY PROCE	8.0	26.0	.5	7.4
FRAME BUFF	16.0	10.0	--	7.4
ANALOG	2.0	8.0	--	7.4
ALU	3.0	5.0	--	7.4
HSFX	3.5	1.5	--	7.4

APPENDIX C

RT-11 SUPPORT

SPOOL Command File

```
!Running SPOOL consist of the following DCL keyboard commands.
SET USR NOSWAP<RET>     !USR must be set noswap to run SPOOL.
LOAD LS:<RET>       !Load the LS handler for use by foreground.
ASSIGN LS: SO0:<RET>!Assign the SPOOL output device.
ASSIGN SP0: LP:<RET>!Assign the SPOOL input device.
FRUN SPOOL/BUFF:256.<RET>
SET USR SWAP<RET>  !Return USR to previous state.
```

VTCOM Command File

```
!To run VTCOM under the FB monitor, make sure nothing is running in
!the foreground (such as SPOOL). Then enter the following DCL
!keyboard commands.
LOAD XL:=F<RET>     !Load the XL: handler for use by the foreground
                    !job.
FRUN VTCOM<RET>     !Run VTCOM in the Foreground.
!To access the utility VTCOM from the background, type control-F.
!This will put the user into the terminal emulation mode.
```

RT-11 Special LP/LS Support

Special software changes have been made to allow RT-11 to support both a serial printer and the parallel device ACT II ink jet printer. The LP: device handler LP.MAC has been modified to allow both the standard parallel printing operation to be performed on the ACT II and a print all mode to be allowed also. Normally the LP: handler intercepts certain control characters and takes certain actions accordingly. This could be avoided by issuing the proper SET command except null characters are still trapped by the handler. The standard LP: handler also sends only 7 of the 8 bits of the ASCII byte. It is necessary to send the full byte to obtain proper operation from the ACT II ink jet printer. The SET LP: PLOT keyboard DCL command puts the LP: handler in the special print all mode of operation by causing a branch to the newly written code that passes all of the information in the data buffer to the output data register regardless of the ASCII character. No character counting or control code interpretation is performed by the handler in this mode and all 8 bits are sent. For normal printing at the ACT II, the keyboard command SET LP: NOPLOT is used. To select the serial printer for output using the PRINT command, the assignment statement ASS LS: LP must be entered. Deassigning LP causes the output from a PRINT command to be output at the parallel printer (ACT II). The file TK.MAC contains the modified version of LP.MAC.

APPENDIX D

IMAGE PROCESSING KEYBOARD COMMANDS

ANSF	Performs adaptive noise smoothing filter.
AVG	Performs frame integration of live video signal into a frame buffer.
BTRLPF	Butterworth Low Pass Filter for FFT.
BTRHPF	Butterworth High Pass Filter.
BTRENH	Butterworth High Pass Enhance.
DISFFT	Displays LOG scaled FFT from memory.
EXPLUT	Exponential LUT.
FFT	Digital 2-D FFT.
GRAD	Performs Gradient operation on an image in a frame buffer.
HISSEG	Calculates intensity segment boundaries according to histogram valleys and statistics.
HISTEQ	Calculates equalized histogram and Loads display LUT.
HISTO	Generates and displays histogram and statistics.
HISTOP	Generates Plot format of histogram.
IFFT	Performs 2-D Inverse FFT.
LINLUT	Loads the display lookup tables with the linear lookup table.
LIVE	Displays Live video image at monitor.
LOGLUT	Logarithmic LUT.
LOGSUB	Performs logarithmic subtraction of two images.
PROFIL	Displays a horizontal profile across an image.
READIP	Reads image data into a frame buffer.
REVERS	Reverses LUT.
ROBERT	Performs Roberts operation on an image.

SCALE	Scales image data to full 8 bits.
SLUT	Loads an S shaped gain (contrast) curve into display LUT.
SOBEL	Performs Sobel edge operation on an image in a frame buffer.
SUBIMG	Subtracts two images.
THRESH	Creates a binary image.
VARY	Creates the variance image.
WNDSMO	Neighborhood smoothed image.
WRITIP	Write image data to a file.

APPENDIX E

HSFX AND DVASS FUNCTIONS

HSFX Subroutines

HFLLUT(table #, array) - Load LUT with values in array.

HFRLUT(table #, array) - Read LUT values into array.

HFLLUT(table #) - Load linear LUT.

HIS(array, sum, maximum occurrence, scale) - Load the histogram result into array, the total number of pixels processed into sum, the maximum occurrence value, and the value to scale the histogram data by to enable a full screen display.

HMXMN(imax, imin) - return the maximum and minimum intensity values in the selected video source.

FNDSFT(shift value) - Return the number of shifts required to scale the data to obtain an 8 bit representation from a 16 bit representation.

I=FSVAL(ival) - return the number of pixels of intensity ival.

I=FGEVAL(ival) - return the number of pixels of intensity greater than or equal to ival.

I=FLSVAL(ival,ix,iy) - return the number of pixels found with intensity ival along with position.

DVASS Subroutines

DVIRCH(channel #, element type) - Define a channel.

SVIRCH(channel #) - select a channel as the current channel.

RVIRRO(row #, data) - Read the row of the current channel into data.

WVIRRO(row #, data) - Write data into the row of the current channel.

RVIRCO(col #, data) - Read the col of the current channel into data.

WVIRCO(col #, data) - Write data into the col of the current channel.

FFTFIL - Generate the right half of the FFT from the left half.

APPENDIX F

ACTPLT USER'S GUIDE

Running ACTPLT

When ACTPLT has been successfully started from the keyboard monitor with a R ACTPLT or ACTPLT entry, an asterisk prompt is displayed after an introductory message. Pressing RETURN or ENTER in response to the asterisk prompt causes the program name, the version number, and the on-line help information to be displayed. Entering a single control-C in response to the asterisk prompt causes ACTPLT to terminate. Information entered in response to the asterisk prompt provided to ACTPLT must conform to the requirements of the Command String Interpreter (CSI) syntax as described in section 1.1 of the "RT-11 Systems Utilities Manual". The program ACTPLT, accepts a command string of up to six image items with their respective options for printing at the ACT II. The options used to specify the format of the printed output currently allow the aspect ratio to be selected, a purge of the ink system prior to creating the copy, the size of the copy to be double or single size, the black and white portion of the printed image to be reversed, and the source of the input data to be taken directly from the frame buffer interface. Creating a copy directly from the frame memory is accomplished by entering a title and including the /S (Screen dump) option switch. The horizontal to vertical aspect ratio is selected with the /A:n option, where n is 2 through 6 selecting the respective aspect ratios 0.850, 1.0, 1.133, 1.275, and 1.417. The ACT II will perform the printing of a purge band on receipt of the first information following a

power-up or reset operation. When several files are to be printed using a single command line, the ink system can be purged prior to the printing of a file with the /P option. The overall size of the copy can be doubled with the use of the /D option switch. Black and white reversal can be selected with the /R option switch. This option causes the plot to be generated with white on the ACT II where the screen is black. The default is no black/white reversal. The other defaults, when no switches are used, select the printing of the copy with an aspect ratio of 1.417 (n=6), no purge band, no double size, and the source of the input data is taken from a .IMG file. When it is necessary to plot many images at a single time, a command file can be created that contains the list of file names and options that would normally be entered at the asterisk prompt. An example of this mode of operation is given below.

```
!PLOTEM.COM - command file to plot a bunch of images.
R ACTPLT
TEST1/A:2/P,TEST2/A:2/R,TEST3,TEST4,TEST5,TEST6
TEST7,TEST8/P,TEST9,TEST10
TEST11
^C
```

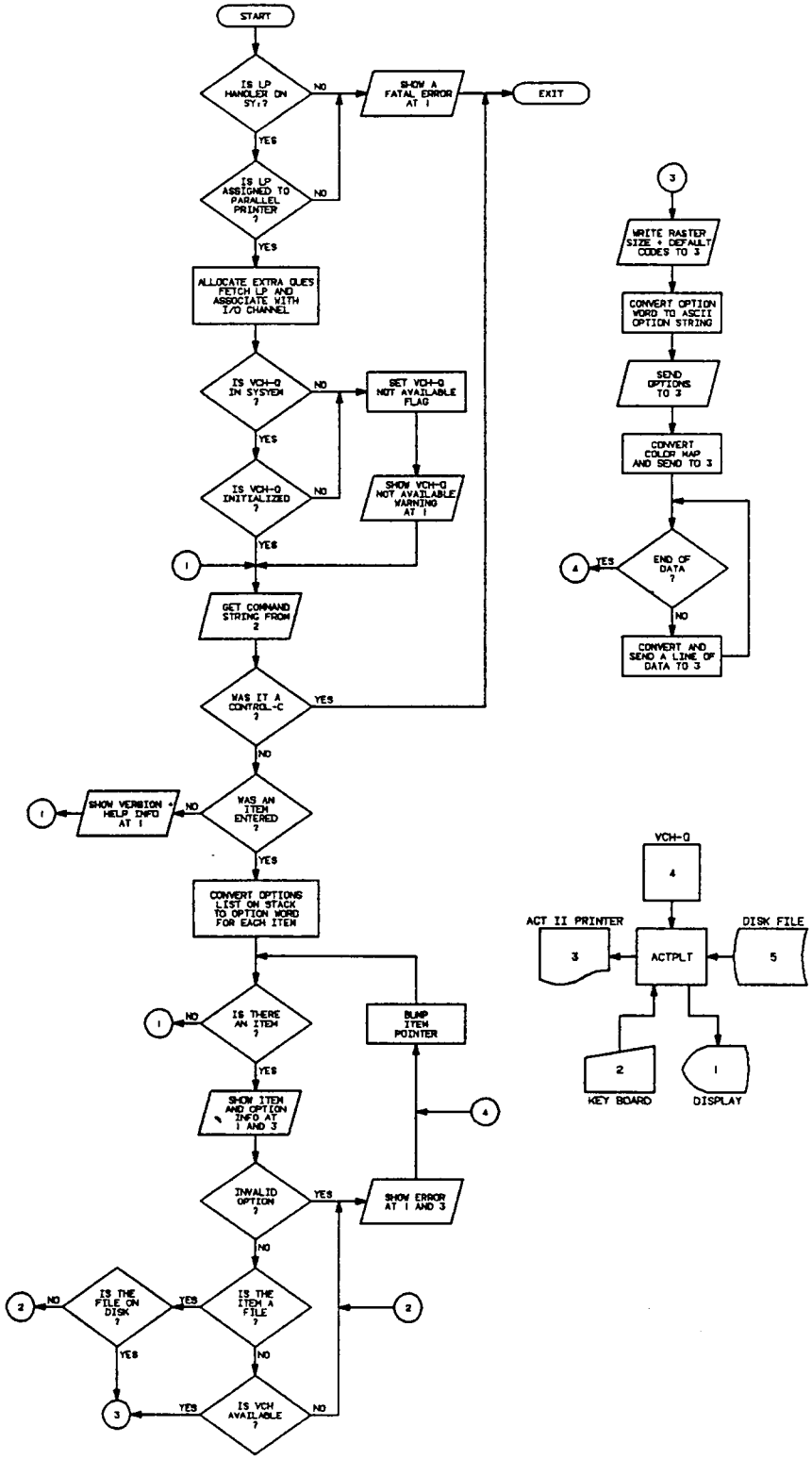


Figure A-1. ACTPLT Flow Chart.

APPENDIX G

SOFTWARE LISTINGS

ACTPLT - Plot Image Utility

ANSF - Adaptive Noise Smoothing Filter

DVASS - Dynamic Virtual Array Storage Routines

EXVAR - Generate Variance image

HSFX - Histogram/Feature Extraction Routines

IPFIO - Image Processor File Input/Output Routines

POLY - Single-Pass Polynomial-Fit-Subtraction Program

```
        .TITLE   ACTPLT          V1.0            19-JAN-85 REV 0
        .SBTTL   DOCUMENTARY
        .REM     *
ACTPLT is a program used to create hardcopy plots from a .IMG file or
directly from FB0 of the IP-512 hardware on the ACT II Ink Jet printer.
It's operation at the keyboard attempts to emulate the operation of a
RT-11 standard utility. ACTPLT accepts the standard command syntax for
items, options, and option values.
        *
        .ENABL   LSB
        .SBTTL   PROGRAMMED REQUESTS AND DEFINITIONS

        .MCALL   .PRINT, .EXIT, .QSET, .WRITW, .WAIT, .DSTATUS
        .MCALL   .RELEA, .CSISP, .LOOKU, .ENTER, .CLOSE
        .MCALL   .READW, .WRITE, .CLOSE,    .FETCH, .TRPSET
;
;       IP-512 Device register and bit definitions
;
APBREG   = 170020
FBBREG   = 170000

LUT.DAT = 0
LUT.ADR = 1
LUT.SEL  = 2

FB.X     = 0
FB.Y     = 2
FB.PAN   = 4
FB.CTRL = 10
FB.DATA = 16

FC.EYC   = 4000
FC.IXARW= 32400
;
; Locations used in System Communication Area (SYSCOM)
;
JSW=     44        ;Job status word
ERRWD=   52        ;Error word
USERRB=  53        ;User Message byte
;
;       I/O parameter definitions
;
OUTCHN=  2         ;Transmit channel no. for LP
INCHN=   3         ;Receive channel no. for data file
BLKS=    4         ;No. of RT11 blocks in buffer (must be 1,2 or 4)
BS=      256.      ;Block size in words
FSIZE=   514.      ;.VCH file size in blocks
FILMAX= 6          ;Maximum number of items per command line.
```

```
;       ACT II Control codes.
;
NOP =    0          ;ASCII null code. No effect on ACTII
LOOK =   1          ;Command to load internal LUT
SIZE =   2          ;Command to specify the picture size
RAS =    3          ;Command to select raster graphics mode
REV =    5          ;Command to turn Black/White reversal on
NOREV =  6          ;Command to acncel Black/white reversal mode
DATA =   7          ;Command indicating the start of a line of data
LF =     10.        ;ASCII Line feed
CR =     13.        ;ASCII carriage return
ONEONE= 21.         ;Command to set the Zoom factor to 1:1
ASPEC = 24.         ;Command used to change the Zoom factor
PBAND = 26.         ;Command used to force the printing of the purge band
DOUBLE= 149.        ;Command to set the Zoom to a factor of 2
;
;       Bits for option select word
;
AOPTB = 400         ;A option bit
POPTB = 1000        ;P option bit
DOPTB = 2000        ;D option bit
ROPTB = 4000        ;R option bit
SOPTB = 10000       ;S option bit
OPTERR= 100000      ;Option error bit
;
;       System Error bits for messages
;
INFOB =  1          ;Information
WARNB =  2          ;Warning
ERRB  =  4          ;Error
FATALB=  10         ;Fatal

        .ENABL   LSB
        .SBTTL   MACROS
;
;       Define some useful Macros for message reporting.
;
$$$=0                       ;Initialize label number
.NLIST   ME
.MACRO   FATAL    STRING            ;Macro for Fatal errors
        CLRB     @#USERRB
        BISB     #FATALB,@#USERRB
        .PRINT   #FATALM
        $$$=$$$+1
        MESSG$   <STRING>,\$$$
        JMP      EXIT
.ENDM
```

```
.MACRO   REMARK   STRING            ;Macro for informational messages
         CLRB     @#USERRB
         BISB     #INFOB,@#USERRB
         .PRINT   #INFOM
         $$$=$$$+1
         MESSG$   <STRING>,\$$$
         .PRINT
.ENDM
.MACRO   WARN     STRING
         CLRB     @#USERRB
         BISB     #WARNB,@#USERRB
         .PRINT   #WARNM
         $$$=$$$+1
         MESSG$   <STRING>,\$$$
         .PRINT
.ENDM
.MACRO   ERROR    STRING
         CLRB     @#USERRB
         BISB     #ERRB,@#USERRB
         .PRINT   #ERRORM
         $$$=$$$+1
         MESSG$   <STRING>,\$$$
         .PRINT
.ENDM
.MACRO   ERRORP   STRING
         CLRB     @#USERRB
         BISB     #ERRB,@#USERRB
         .PRINT   #ERRORM
         $$$=$$$+1
         MESSG$   <STRING>,\$$$
         .PRINT
         .WRITE   #WAREA,#OUTCHN,#NOTPRO,#28.,#0
         .WRITE   #WAREA,#OUTCHN,#LFS,#2,#0
.ENDM
.MACRO   MESSG$   STRING,LABEL
         MOV      #L'LABEL,R0
.PSECT   $$TEXT,D
L'LABEL: .ASCIZ   /STRING/
.PSECT
.ENDM
         .ENABL   LSB
         .SBTTL   CHECK RESOURCES

ACTPLT:
;
;        Check system resources.
;
         .DSTATU  #DSAVE,#OFILE
         BCS      10$
         CMPB     DSAVE,#3
         BEQ      RITELP
```

```
        FATAL   <LP not assigned properly - Do @TEKLP>
10$:    FATAL   <LP not on system or not installed>
RITELP: .QSET   #XQUE,#4 ; Allocate four (4) extra QUE elements for
                ; I/O in JOB space.
        .FETCH  LIMIT,#OFILE        ; Pull LP handler into memory for
                        ; use by the program.
        BCC     15$
        FATAL   <FETCH on LP failed>
15$:    MOV     R0,LIMIT
        .ENTER  #EAREA,#OUTCHN,#OFILE
        BCC     20$
        FATAL   <ENTER on LP failed>
20$:    .TRPSET #TAREA,#TRPADR      ; Next see if IP-512 is installed.
        TST     @#APBREG
        TST     NOIP
        BEQ     CSIIN
        WARN    <IP-512 not installed on this system>
;
        .ENABL  LSB
        .SBTTL  GET INPUT STRING
;
CSIIN:  REMARK  <Plot your favorite image on the ACT II !>
        MOV     SP,SPSAV ;SAVE SP ACROSS CALL
        .CSISP  #IBLK,#ITYP,#0,#INSTRN
        BCC     45$
CSIERR: MOV     SPSAV,SP
        TSTB    @#ERRWD
        BNE     40$
        ERROR   <CSI error, Invalid command line - Retry>
        BR      CSIIN
40$:    ERROR   <CSI error, Invalid device - Retry>
        BR      CSIIN
45$:    TST     IFILE               ; Was a file entered?
        BNE     OPTION
        MOV     SPSAV,SP
        .PRINT  #HELP
        BR      CSIIN
;
        .ENABL  LSB
        .SBTTL  PROCESS OPTIONS
        .REM    *
```

The above .CSISP has placed the options for each of the items on the stack. This section of codes checks the items on the stack and sets the appropriate bit in the option select word (FILOPT)for each item. The /A value is placed in the low byte of the option select word. Bad options and a bad option A value cause the error bit in the option word to be set. Values for any of the other option switches are ignored. Two conditions can exists that will cause .CSISP to be reissued with an error message at this point. When an input string contains an output file or if the maximum number of files is exceeded. The format of the options on the stack is as follows:

```
                        word
SP after .CSISP -->      1      The number of options on the stack
                         2      Low Byte - The ASCII character of the option
                                Bits 8-14  The file number to associate the
                                option.
                                Bit 15   = 1 If the option has a value
                         3      If Bit 15 was set in word 2, this word is
                                the option's value. Else, it would be the
                                same as word 2 for the next option.
                                etc.
                    *
;
OPTION:  MOV     #FILOPT,R0          ;Point to start of file option area.
         MOV     #FILMAX,R1          ;There is a maximum number of 6.
10$:     CLR     (R0)+               ;Clear file option area.
         DEC     R1
         BNE     10$
         MOV     (SP)+,R0 ;Get number of options in CS and bump stack
         BEQ     NOOPTS              ;branch if no options
NXOPT:   MOV     (SP),R1             ;Get option
         MOV     R1,R2               ;Make a copy.
         SWAB    R1                  ;Put item number in low byte
         BIC     #^C<17>,R1          ;Clear all but first 4 bits
         SUB     #3,R1               ;Allow for output spec numbers
         BLT     CSIERR              ;ERROR if output spec was entered.
         ASL     R1                  ;Make item number a word offset in
                                     ;option area
         ADD     #FILOPT,R1          ;Point to option area for file.
         CMP     R1,#OPEND           ;Check to see if out of option area.
         BGT     CSIERR
;
;        Convert option into appropriate bit in option word.
;
         CMPB    R2,#'A              ;Is it an A switch?
         BNE     120$
         TST     R2
         BPL     165$     ;A option must have a value else error.
         MOV     2(SP),R3
         CMP     R3,#2               ;A option value must be 2,3,5, or 6
         BLT     165$
         CMP     R3,#4
         BEQ     165$
         CMP     #6,R3
         BLT     165$
         MOVB    R3,(R1)  ;If okay mov into low byte of opt word.
         BIS     #AOPTB,(R1)         ;Show A option a was selected.
         BR      180$
120$:    CMPB    R2,#'P              ;Is it a P switch?
         BNE     150$
         BIS     #POPTB,(R1)
         BR      180$
```

```
150$:   CMPB    R2,#'D          ;Is it a D switch?
        BNE     155$
        BIS     #DOPTB,(R1)
        BR      180$
155$:   CMPB    R2,#'R          ;Is it an R switch?
        BNE     160$
        BIS     #ROPTB,(R1)
        BR      180$
160$:   CMPB    R2,#'S          ;Is it an S switch?
        BNE     165$
        BIS     #SOPTB,(R1)
        BR      180$
165$:   BIS     #OPTERR,(R1)
180$:   TST     (SP)+           ;Bump SP to next option.
        BPL     200$            ;If it had a Value
190$:   TST     (SP)+           ;Bump again.
200$:   DEC     R0              ;Decrement number of options.
        BNE     NXOPT           ;Go get next option if there is one.
NOOPTS: MOV     SPSAV,SP ;Restore SP as it was before .CSISP
;
        .ENABL  LSB
        .SBTTL  PREPARE FOR PLOT
;
        MOV     #INSTRN,STRNPT          ;Get input string pointer
        MOV     #-1,FILCNT              ;Initialize file counter.
LOOP:   INC     FILCNT
        MOV     FILCNT,R2
        ASH     #3,R2           ;Make a word offset.
        ADD     #IFILE,R2       ;Point to an item.
        TST     (R2)            ;Is there one there?
        BNE     ANITEM
        JMP     CSIIN           ;Go get next string.
ANITEM: .PRINT  #CRLF
        CLR     SOURCE          ;Reset source of input flag
        MOV     FILCNT,R3       ;Create a pointer to option word.
        ASL     R3
        ADD     #FILOPT,R3
        BIT     #SOPTB,(R3)     ;Test for source of input.
        BEQ     AFILE
        MOV     #2,SOURCE       ;Show source is from screen.
AFILE:  MOV     #SOUR,R2
        ADD     SOURCE,R2
        MOV     (R2),R0
        .PRINT
```

```
        MOV     STRNPT,R1
        MOV     R1,R0
        CLR     R2
CHKSTR: CMPB    (R1),#',
        BEQ     111$
        CMPB    (R1),#NOP
        BEQ     111$
        INC     R1
        INC     R2
        BR      CHKSTR
111$:   MOVB    #200,(R1)+
        MOV     R1,STRNPT       ;Reset Pointer at end of processed
                                ;characters
        INC     R2
        ASR     R2
        MOV     R0,STRADR       ;Save String start address
        MOV     R2,STRCNT       ;and length for later.
        .PRINT
        .PRINT  #DASHES
;
; Write file name of file to be processed to ACT II
;
        .WRITE  #WAREA,#OUTCHN,STRADR,STRCNT,#0
;
; Now check for bad file file options
;
        MOV     FILCNT,R3
        ASL     R3
        ADD     #FILOPT,R3
        TST     (R3)
        BPL     CHKFIL
        ERRORP  <Unallowed file option>
        BR      LOOP
CHKFIL: TST     SOURCE
        BEQ     ISFILE
        JMP     LOOP
ISFILE: CALL    IOPEN
        BCC     SNDSET
        JMP     LOOP
SNDSET: .WRITE  #WAREA,#OUTCHN,#LFS,#2,#0
        .WRITE  #WAREA,#OUTCHN,#SETUP,#5,#0         ;Write Raster and
                                                    ;size code, and defaults.
;
; Convert Option word for this item into the codes to be sent to ACT II
;
        TST     (R3)
        BEQ     PLOTIT          ;Branch if no options
        CLR     R2
        MOV     #OPTSEL,R1
```

```
DOPT:   BIT     #DOPTB,(R3)
        BEQ     POPT
        MOV     #DOUBLE,(R1)+
        INC     R2
POPT:   BIT     #POPTB,(R3)
        BEQ     AOPT
        MOV     #PBAND,(R1)+
        INC     R2
AOPT:   BIT     #AOPTB,(R3)
        BEQ     ROPT
        MOVB    #ASPEC,(R1)+
        MOVB    (R3),(R1)+
        INC     R2
ROPT:   BIT     #ROPTB,(R3)
        BEQ     OPTS
        MOV     #REV,(R1)
        INC     R2
OPTS:   .WRITW  #WAREA,#OUTCHN,#OPTSEL,R2,#0
;
PLOTIT: .PRINT  #CRLF     ; Dispatch to correct plotting routine based
                          ; on SOURCE
        MOV     #PLOT,R0
        ADD     SOURCE,R0
        JMP     @(R0)
;
;Trap intercept routine used when checking if IP hardware is present
;
TRPADR: MOV     #1,NOIP
        RTI
EXIT:   .PRINT
        MOV     #1,R0             ;Set R0=1 for normal abort.
        .EXIT
;
        .ENABL  LSB
        .SBTTL  PLOT FROM .VCH FILE
        .REM    *
        UNLOAD COLOR MAP
This section of code gets the color map from the AP-512 as consecutive
Red, Green, and Blue Bytes.       *

PLOTHF: MOV     #2,R5             ; Set to read in first 2 blocks
        CALL    READB             ; Read block 0-1 from file (color
                          ; map)
        MOV     #256.,R0          ; Number of values in color map.
        MOV     #$BFR,R1 ;Pointer to IMG color map just read in.
        MOV     #CLOOK,R2         ;Pointer to ACT color map area.
10$:    MOVB    (R1)+,(R2)+       ;Get a IMG RED value.
        MOVB    (R1)+,(R2)+
        MOVB    (R1)+,(R2)+
        DEC     R0                ;See if done with color map.
        BNE     10$               ;No, go get new value.
```

```
;Write LUT code and color map to ACT II
;
        .WRITE   #WAREA,#OUTCHN,#LUT,#386.,#0
;
;       Now get data
;
        .REM     *
The data is stored as bytes and range from 0 to 255. The colors for
these values are mapped using the LUT. Each block read from the file
contains a horizontal line of the picture image (512 pixels).
                 *
REREAD: MOV      #1,R5              ;Read in one block of data.
        CALL     READB
        BCS      40$                ;READB sets carry if EOF
        MOV      #OUTA,R2 ;Storage location for output buffer for ACT
                                    ;II
        MOV      #$BFR,R1 ;Buffer holding a line of data from file.
        MOV      #256.,R4 ;
30$:    MOV      (R1)+,(R2)+
        DEC      R4
        BNE      30$
        .WAIT    #OUTCHN
        .WRITE   #WAREAA,#OUTCHN,#DATAA,#257.,#0
        BR       REREAD
40$:    .CLOSE   #INCHN
        JMP      LOOP
;
        .ENABL   LSB
        .SBTTL   PLOT FROM IP FRAME BUFFER

PLOTH:  MOV      #APBREG,R5         ;Get address of AP register.
        MOV      #256.,R4
        MOV      #CLOOK,R2
        CLR      R1
1$:     MOV      #3,R0     ;Number of values in color map.
        CLR      LUT.SEL(R5)
2$:     ADD      #40000,LUT.SEL(R5)
        MOVB     R1,LUT.ADR(R5)
        MOVB     LUT.DAT(R5),(R2)+
        SOB      R0,2$
        INC      R1
        SOB      R4,1$
;
; Write LUT code and color map.
;
        .WRITE   #WAREA,#OUTCHN,#LUT,#386.,#0
;
; Now get data
;
        MOV      #FBBREG,R1
        BIC      #FC.EYC,FB.CTRL(R1)
```

```
        BIS     #FC.IXARW,FB.CTRL(R1)
        CLR     FB.Y(R1)
        MOV     FB.PAN(R1),FB.X(R1)
        MOV     #512.,R3
30$:    MOV     #OUTA,R2
        MOV     #512.,R4
40$:    MOVB    FB.DATA(R1),(R2)+
        SOB     R4,40$
        .WAIT   #OUTCHN
        .WRITE  #WAREAA,#OUTCHN,#DATAA,#257.,#0
        INC     FB.Y(R1)
        SOB     R3,30$
        JMP     LOOP
;
;*****************************************************
; *** IOPEN ***            OPEN A SINGLE INPUT FILE
        .SBTTL  OPEN A FILE
;
; All error messages are handeled internally and carry bit is set.

        .ENABL  LSB
IOPEN:: MOV     FILCNT,R4
        ASH     #3,R4
        ADD     #IFILE,R4
        CLR     IEOF            ; Reset EOF flag
        CLR     IREC
        .FETCH  LIMIT,R4        ; Fetch handler if necessary
        BCC     40$
        ERRORP  <FETCH on file failed, Bad device name>
        SEC
        RETURN
40$:    MOV     R0,LIMIT        ; Update free space pointer
        .LOOKU  #LAREA,#INCHN,R4  ; Find the file
        BCC     50$
        ERRORP  <File not found>
        .CLOSE  #INCHN   SEC
        RETURN
50$:    CMP     R0,#FSIZE
        BEQ     60$
        ERRORP  <File not proper IMG size>
        .CLOSE  #INCHN
        SEC
60$:    RETURN
;
;*****************************************************
        .SBTTL  READ FROM STORAGE
        .ENABL  LSB
; Read a buffer load of data from the input file
READB:
        TST     IEOF            ; EOF yet?
        BNE     105$
```

```
        MOV     R5,-(SP)        ; Save block count
        SWAB    R5              ; Times 256 words
        MOV     R5,ISIZE        ; Setup word count
        .READW  #IAREA
        BCS     100$            ; CS= input error or EOF
        ADD     (SP)+,IREC      ; Bumpt to next block
        RETURN

100$:   TST     (SP)+
        TSTB    @#ERRWD
        BNE     110$            ; NE= not EOF, so FATAL
        INC     IEOF            ; Set EOF flag
105$:   SEC                     ; Signal EOF on input
        RETURN
110$:   .CLOSE  #INCHN   FATAL  <READ File hard error>
        RETURN
IAREA:  .BYTE   INCHN,10        ; Input EMT area, FCN 10 (READW)
IREC:   .WORD   0               ; Record No.
        .WORD   $BFR
ISIZE:  .WORD   0               ; Size of transfer in blocks
        .WORD   0               ; Wait
IEOF:   .WORD   0               ;.NE. 0 if EOF
;
        .SBTTL  DATA STORAGE AND PARAMETERS
; File names from .CSISPC
;
IBLK:   .BLKW   15.             ;3 Output files, 5 words each (NU)
IFILE:  .BLKW   24.             ;6 Input files (#3-8), 4 words each.
ITYP:   .RAD50  *VCH*           ;Default extension for input files.
        .WORD   0,0,0
INSTRN: .BLKB   82.             ;Buffer to hold input string
; General storage
;
SPSAV:  .WORD   0               ;Location to save stack pointer
FILOPT: .BLKW   5               ;Area for file option words
OPEND:  .WORD   0               ;End of file option area
FILCNT: .WORD   0               ;Location to hold current item count
STRNPT: .WORD   0               ;Address of current item in INSTRN
SOUR:   SOURC0                  ;Address of input source messages
        SOURC2
DSAVE:  .BLKW   4               ;Area for .DSTATUS on LP
OFILE:  .RAD50  *LP *           ;LP value
        .WORD   0,0,0
$BFR::  .BLKW   BS*BLKS         ;Area to receive input from a file.
XQUE:   .BLKW   40.             ;QUE area for 4 extra elements.
NOIP:   .WORD   0               ;Word to indicate if IP is present.
PLOT:   PLOTHF                  ;Addresses to ploting routines
        PLOTH
SOURCE: .WORD   0               ;Indicator of source 2=from VCH-Q
STRADR: .WORD   0
STRCNT: .WORD   0
```

```
; Command structures for ACT II
;
LUT:      .BYTE    NOP,LOOK,0,1        ;LUT is 256
CLOOK:    .BLKB    768.                ;ACT color map area.
DATAA:    .BYTE    NOP,DATA            ;Data code.
OUTA:     .BLKW    256.                ;ACT data area for 1 line.
LFS:      .BYTE    10.,10.,10.,10.
SETUP:    .BYTE    RAS,SIZE,0,2,0,2    ;Raster,size 512X512
DEFAUL:   .BYTE    ASPEC,6,ONEONE,NOREV
OPTSEL:   .BLKB    8.
;         EMT areas
;
LAREA:    .BLKW    3
WAREA:    .BLKW    5
WAREAA:   .BLKW    5
WAREAB:   .BLKW    5
EAREA:    .BLKW    4
TAREA:    .BLKW    2                   ;EMT area for .TRPSET
          .LIMIT              ;Area to recieve job limits at link time.
LIMIT=    .-2                 ;Pointer to job high limit.
;
          .PSECT   $$TEXT
HELP:     .ASCII   /ACTPLT V05.01/<15><12>
          .ASCII   <15><12>/ACTPLT Query Information/<15><12>
          .ASCII   /Maximum of 6 files per command line/<15><12>
          .ASCII   */A:n  Select aspect ratio*<15><12>
          .ASCII   */P    Purge ink before printing file*<15><12>
          .ASCII   */D    Print at double size*<15><12>
          .ASCII   */R    Selects Black/White reversal*<15><12>
          .ASCII   */S    Selects printing image from FB0*<15><12>
          .ASCII   <15><12>*Defaults A:6,No purge,No double,No Reversal,*
          .ASCII   *From file*<15><12><15><12>
          .ASCII   *Example string: FILEA/A:2/P,FILEA/A:6,FILEB/A:2/D*
          .ASCIZ   <15><12>/Control-C to abort ACTPLT/
SOURC0:   .ASCII   /?ACTPLT-I-Processing .IMG file: /<200>
SOURC2:   .ASCII   /?ACTPLT-I-Processing  IP-FB0: /<200>
FATALM:   .ASCII   /?ACTPLT-F-/<200>
INFOM:    .ASCII   /?ACTPLT-I-/<200>
WARNM:    .ASCII   /?ACTPLT-W-/<200>
ERRORM:   .ASCII   /?ACTPLT-E-/<200>
CRLF:     .ASCII   <15><12><200>
DASHES:   .ASCII   / -- /<200>
NOTPRO:   .ASCII   / *** NOT PROCESSED - ERROR ON FILE ****/
          .PSECT
          .END     ACTPLT
```

```
      PROGRAM ANSF
C
C Program to perform adaptive noise smoothing filter using both the
C image processor and array processor.  The program prompts for the
C noise variance value.
C
      INTEGER  LKPAN,TEMPL(14,4),ILINE(512)
      REAL     V(512),VPS(512),NVAR
C
C Setup image processor and frame buffers
C
      CALL SELGRP(1)
      CALL SELVUE(0)
      ISPN=LKPAN(0)
      CALL SETSPN(0,(ISPN.AND."777))
      ISPN=LKPAN(1)
      CALL SETSPN(1,(ISPN.AND."777))
      ISPN=LKPAN(2)
      CALL SETSPN(2,(ISPN.AND."777))
      CALL SETADM(1)
C
C Initialize Array procesor work space.
C
      CALL VINIT(TEMPL,4)
C
      WRITE(5,1)
1     FORMAT(1X,'ENTER NOISE VARIANCE:',$)
      READ(5,2)NVAR
2     FORMAT(F10.0)
C
C Calculate Local Mean And Variance images.
C
      CALL EXVAR
C
C Loop to convert variance image to weighting factor K=V/(V+NVAR)
C
      DO 10 I=1,480
C
      IY=I-1
      CALL READHL(2,0,IY,512,ILINE)
      CALL VI2SP(ILINE,1,V,1,512,'R')
      CALL VSADDW(RVAR,V,1,VPS,1,512,'R')
      CALL VDIVW(V,1,VPS,1,V,1,512)
      CALL VSMULW(255.,V,1,V,1,512,'R')
      CALL VSP2I(V,1,ILINE,1,512,'R')
      CALL VWAIT
      CALL WRITHL(2,0,IY,512,ILINE)
C
10    CONTINUE
```

```
C
C To reconstruct the image First calculate K*I
C
        CALL SETK(1,0)
        CALL SETK(2,0)
        CALL SETK(3,255)
        CALL SETOPC(3,0)
        CALL SETMUL(1,0,0,0)
        CALL SETSFT(1,1,8,0)
        CALL SUALFN(0,2,0,4,1,3,-8,0,0,0,0,0,0)
        CALL DOALFN(1,1,0,0,0)                !K*I into FB0
C
C Now calculate local mean weighting factor 1-K
C
        CALL SETK(3,255)
        CALL SETOPC(2,1)
        CALL SUALFN(4,4,2,4,0,2,0,0,0,0,0,2)
        CALL DOALFN(1,0,0,1,0)                !1-K into FB2
C
C Calculate (1-K)M
C
        CALL SETOPC(3,0)
        CALL SUALFN(1,2,1,4,1,3,-8,0,0,0,0,1)
        CALL DOALFN(1,0,1,0,0)                !(1-K)M into FB1
C
C Now sum weighted Local Mean And Variance, IR=(1-K)M + K*I
C
        CALL SUALFN(0,4,1,4,0,3,0,0,0,0,1,2)
        CALL DOALFN(1,0,0,1,0)      !Reconstructed Image into FB2
C
        CALL VDONE
C
        STOP
        END
```

```
        .TITLE   DYNAMIC VIRTUAL ARRAY STORAGE SOFTWARE
        .IDENT   /V01.00/
        .NLIST   SEQ,LOC,BIN,SYM
        .SBTTL   DOCUMENTARY
        .REM     *
```

This software contains the modules that allow virtual memory to be used as storage for large arrays. In particular, arrays of 512X512 size of either byte, integer, real, or complex elements. The software is composed of four parts; array location management, direct frame buffer transfers, line or column access, and FFT completion.

Array Location Management-The arrays in virtual memory can be allocated dynamically by the user software. The array location and type are kept by the software in a table. When an array type is defined (DEFVCH), the software determines if it will fit in the available memory, stores the location and type in a table, and returns a channel number that the user software uses to reference that array. When a particular array in virtual memory is to be accessed, the channel of that array is selected for use (SELVCH). That elements of that array can then be accessed using the line, column, or element access routines. Channels can be deleted from the table to free up memory space (DELVCH).

DEFine Virtual CHannel
CALL DEFVCH(I_Aarray_Type,I_channel_#)

I_Array_Type	0 = byte 1 = integer 2 = real 3 = complex
I_Channel_#	The returned channel to reference the defined array.

SELect Virtual CHannel
CALL SELVCH(I_Channel#)

I_Channel_#	Channel of array to access.

DELete Virtual CHannel
CALL DELVCH(I_Channel#)

I_Channel_#	Channel of array to delete.

Direct Frame Buffer Transfers-The routines in this function allow a Frame buffer to be copied into a defined and selected virtual memory array.

Integer Frame Buffer To Virtual Array
CALL IFBTVA(I_Frame_Buffer)

Virtual Array To Integer Frame Buffer
CALL VATIFB(I_Frame_Buffer)

Byte Frame Buffer To Virtual Array
CALL BFBTVA(I_Frame_Buffer)

```
Virtual Array To Byte Frame Buffer
CALL VATBFB(I_Frame_Buffer)

        Element Access-Routines exist to access the defined and
selected elements in the array by either Rows (Horizontal video lines)
or columns (vertical video lines.)

Read Virtual Array ROW
CALL RVAROW(I_Row_#, Data_Type_linear_Array)

Write Virtual Array ROW
CALL WVAROW(I_Row_#, Data_Type_linear_Array)

Read Virtual Array Column
CALL RVACOL(I_Col_#, Data_Type_linear_Array)

Write Virtual Array COLumn
CALL WVACOL(I_Col_#, Data_Type_linear_Array)

FFT Fill Routine
CALL FFTFIL

        *
;+
;  Include Frame buffer definitions and macros
;-
        .INCLUDE /DK:ITEHED.MAC/
;
        .PAGE
        .SBTTL   ADDRESSES AND DEFINITIONS
;
MMSR0   =       177572          ; Memory Status Register 0
MMSR3   =       172516          ; Memory Status Register 3
UISAR0  =       177640          ; User Page Address Register 0
UISAR1  =       177642          ; User Page Address Register 1
PSW     =       177776          ; Processor Status Word
PUMODE  =       30000           ; Previous Mode = User Bits
ADRS22  =       20              ; Enable 22 bit addressing
SYSPTR  =       54              ; Location of base of RMON
$MEMPT  =       430             ; Offset to memory configuration
block
$RAMSZ  =       -2              ; Offset to Total Physical memory
;
FC.IXARW=FC.SAR!FC.SAW!FC.AXD!FC.EXC
;
```

```
        .SBTTL  MACRO DEFINITIONS
;+
;  FBSET - Sets up common needs for transfers to or from Frame Buffer
;-
        .MACRO  FBSET
        TST     IFLG
        BNE     1$
        JSR     PC,INIT
1$:     FBDBA   2,R1
        MOV     @R1,R1              ; Get base address of frame buffer
        BIC     #FC.EYC,FB.CTRL(R1)         ; Make sure auto y inc is
                                            ; off
        BIS     #FC.IXARW,FB.CTRL(R1)       ; Auto inc x on read or
                        ; write
        CLR     FB.Y(R1)                    ; Start at y=0
        MOV     CHNSTR,@#UISAR0             ; Get starting address
        BIS     #1,@#MMSR0                  ; Enable MMU
        MOV     R1,R2
        ADD     #FB.DATA,R2
        .ENDM
;;
        .MACRO  ENDTRN
        BIC     #1,@#MMSR0                  ; Disable MMU
        RTS     PC                          ; Return to caller
        .ENDM
;
        .SBTTL  MEMORY SIZING ROUTINE
;
MEMINI: MOV     @#SYSPTR,R1         ; Get monitor base
        MOV     $MEMPT(R1),R0       ; Get offset to memory control block
                                    ; ptr
        ADD     R1,R0               ; make address
        MOV     $RAMSZ(R0),R1       ; point to total memory on system in
                                    ; 32w blocks
        ASH     #-4,R1              ; convert to KB
        BIC     #170000,R1
        SUB     #56.,R1             ; adjust for first 28K words
;
; Now determine how many 256 KB buffers
;
        ASH     #-8.,R1
        MOV     R1,NOCHS ; Store for later
        MOV     #1,MEMFLG           ; Show that we have done this
        RTS     PC
;
        .SBTTL  PREVIOUS MODE IS USER
;
INIT:   BIS     #PUMODE,@#PSW       ; Show previuos mode is user in PSW
        BIS     #ADRS22,@#MMSR3     ; Enable 22-bit addressing
        MOV     #1,IFLG             ; Show that this has been done
        RTS     PC
```

```
        .SBTTL   DEFINE VIRTUAL CHANNEL
;+
;  DEFine Virtual CHannel
;  CALL DEFVCH(I_transfer_type,I_channel_ #)
;-
DEFVCH:: TST     MEMFLG             ; Has memory been sized?
         BNE     3$
         JSR     PC,MEMINI          ; Then go do it
3$:      TST     (R5)+
         MOV     @(R5)+,R1          ; get type of transfer        ;
(byte,integer,..)
         MOV     #1,R0
         ASH     R1,R0              ; convert to number of channels
         ; needed
         MOV     CHUSE,R2 ; Get channel use bit map
         MOV     #1,R3
         ASH     R0,R3
         DEC     R3                 ; Fit mask
         MOV     #16.,R4
         SUB     R0,R4              ; allowable shifts for mask
         CLR     TEMP
5$:      BIT     R3,R2
         BEQ     FIT
         ASL     R3
         INC     TEMP
         SOB     R4,5$
         MOV     #-1,@(R5)          ; Show no fit
         RTS     PC
FIT:     BIS     R3,CHUSE
         MOV     TEMP,R3
         MOV     R3,@(R5) ; Return selected channel
         MOV     R1,R0
         MOV     R3,R1
         MUL     BOFF,R1
         ADD     #1600,R1
         ASH     #2,R3              ; make a double word offset
         ADD     #CHNO,R3 ; R3 points to channel info area
         MOV     R1,(R3)+ ; Set starting page address
         MOV     R0,(R3)            ; Set transfer type
         RTS     PC
```

```
        .SBTTL   DELETE VIRTUAL CHANNEL
;+
;  DELete Virtual CHannel
;  CALL DELVCH(I_Channel#)
;-
DELVCH:: TST     (R5)+
         MOV     @(R5),R0 ; get channel number to select
         MOV     R0,R1              ; save it
         ASH     #2,R0              ; make it a double word offset
         ADD     #CHN0,R0
         TST     (R0)+
         BEQ     20$                ; Invalid Channel
         MOV     #1,R2
         ASH     (R0),R2            ; Get Transfer type
         MOV     #1,R3
         ASH     R2,R3
         DEC     R3
         ASH     R1,R3
         BIC     R3,CHUSE
         CLR     (R0)
         CLR     -(R0)
20$:     RTS     PC
;
        .SBTTL   SELECT VIRTUAL CHANNEL
;+
;  SELect Virtual CHannel
;  CALL SELVCH(I_Channel#)
;-
SELVCH:: TST     (R5)+
         MOV     @(R5),R0 ; get channel number to select
         ASH     #2,R0              ; make it a double word offset
         ADD     #CHN0,R0
         MOV     (R0)+,CHNSTR
         MOV     (R0),R1            ; get transfer type
         MOV     #4,TWORDS
         CMP     #3,R1              ; see if complex
         BEQ     1$
         MOV     R1,TWORDS
1$:      MOV     R1,TRTYPE
         MOV     R1,R2
         ADD     #3,R2
         MOV     R2,SHIFTS
         MOV     XFER,R2
         ASH     R1,R2
         MOV     R2,TRANSF
         RTS     PC
```

```
;
        .PAGE
        .SBTTL   READ/WRITE INTEGER FRAME BUFFER
;+
;  Integer Frame Buffer To Virtual Array
;  CALL IFBTVA(I_Frame_Buffer)
;-
IFBTVA:: FBSET
        MOV     FB.PAN(R1),FB.X(R1)         ; Set pan to current x
        MOV     #512.,R4
4$:     CLR     R0
        MOV     #512.,R3
5$:     MOV     (R2),-(SP)                  ; Put data on stack
        MTPI    (R0)+                       ; data from stack to vir
                                        mem
        SOB     R3,5$
        INC     FB.Y(R1)                    ; Next line
        ADD     #20,@#UISAR0                ; Bump page base address by
                                            ; 512W
        SOB     R4,4$
        ENDTRN
;+
;  Virtual Array To Integer Frame Buffer
;  CALL VATIFB(I_Frame_Buffer)
;-
VATIFB::
        FBSET
        CLR     FB.PAN(R1)
        CLR     FB.SCROLL(R1)
        CLR     FB.X(R1)
        MOV     #512.,R4
4$:     CLR     R0
        MOV     #512.,R3
5$:     MFPI    (R0)+                       ; Put data on stack
        MOV     (SP)+,(R2)                  ; data from stack to FB
        SOB     R3,5$
        INC     FB.Y(R1)                    ; Next line
        ADD     #20,@#UISAR0                ; Bump page base address by
                                            ; 512W
        SOB     R4,4$
        ENDTRN
```

```
        .SBTTL   READ/WRITE BYTE FRAME BUFFER
;+
;  Byte Frame Buffer To Virtual Array
;  CALL BFBTVA(I_Frame_Buffer)
;-
BFBTVA:: FBSET
         MOV      FB.PAN(R1),FB.X(R1)
         MOV      #512.,R4
4$:      CLR      R0
         MOV      #256.,R3
5$:      MOV      (R2),-(SP)
         MOV      (R2),R5
         SWAB     R5
         BIS      R5,(SP)
         MTPI     (R0)+
         SOB      R3,5$
         INC      FB.Y(R1)
         ADD      #10,@#UISAR0
         SOB      R4,4$
         ENDTRN
;+
;  Virtual Array To Byte Frame Buffer
;  CALL VATBFB(I_Frame_Buffer)
;-
VRTFBB:: FBSET
         CLR      FB.SCROLL(R1)
         CLR      FB.PAN(R1)
         CLR      FB.X(R1)
         MOV      #512.,R4
4$:      CLR      R0
         MOV      #256.,R3
5$:      MFPI     (R0)+
         MOV      (SP),(R2)
         SWAB     (SP)
         MOV      (SP)+,(R2)
         SOB      R3,5$
         INC      FB.Y(R1)
         ADD      #10,@#UISAR0
         SOB      R4,4$
         ENDTRN
```

```
        .SBTTL   READ/WRITE VIRTUAL ARRAY BY ROW
;+
;  Read Virtual Array ROW
;  CALL RVAROW(I_Row_#, Data_Type_linear_Array)
;-
RVAROW:: TST      IFLG
         BNE      1$
         JSR      PC,INIT
1$:      MOV      @2(R5),R0                ; get the line number
         ASH      SHIFTS,R0
         ADD      CHNSTR,R0
         MOV      R0,@#UISAR0
         MOV      4(R5),R3
         MOV      TRANSF,R0
         CLR      R1
         BIS      #1,@#MMSR0
5$:      MFPI     (R1)+
         MOV      (SP)+,(R3)+
         SOB      R0,5$
         ENDTRN
;
;+
;  Write Virtual Array ROW
;  CALL WVAROW(I_Row_#, Data_Type_linear_Array)
;-
WVAROW:: TST      IFLG
         BNE      1$
         JSR      PC,INIT
1$:      MOV      @2(R5),R0                ; get the line number
         ASH      SHIFTS,R0
         ADD      CHNSTR,R0
         MOV      R0,@#UISAR0
         MOV      4(R5),R3
         MOV      TRANSF,R0
         CLR      R1
         BIS      #1,@#MMSR0
5$:      MOV      (R3)+,-(SP)
         MTPI     (R1)+
         SOB      R0,5$
         ENDTRN
```

```
        .SBTTL   READ/WRITE VIRTUAL ARRAY BY COLUMN
;+
; Read Virtual Array Column
; CALL RVACOL(I_Col_#, Data_Type_linear_Array)
;-
RVACOL:: TST      IFLG
         BNE      1$
         JSR      PC,INIT
1$:      MOV      @2(R5),R1          ; get column number
         MOV      4(R5),R3 ; get address of buffer
         ASH      TRTYPE,R1          ; make it an element address
         MOV      #10,R0
         ASH      TRTYPE,R0          ; R0 contains amount to point to
                  ; next row
         MOV      CHNSTR,@#UISAR0    ; start at first row
         MOV      #512.,R2
         BIS      #1,@#MMSR0
         MOV      R1,-(SP)
5$:      MOV      TWORDS,R4
10$:     MFPI     (R1)+
         MOV      (SP)+,(R3)+
         SOB      R4,10$
         MOV      (SP),R1
         ADD      R0,@#UISAR0
         SOB      R2,5$
         TST      (SP)+
         BIC      #1,@#MMSR0
         RTS      PC
;+
; Write Virtual Array COLumn
; CALL WVACOL(I_Col_#, Data_Type_linear_Array)
;-
WVACOL::
         TST      IFLG
         BNE      1$
         JSR      PC,INIT
1$:      MOV      @2(R5),R1          ; get column number
         MOV      4(R5),R3 ; get address of buffer
         ASH      TRTYPE,R1          ; make it an  element address
         MOV      #10,R0
         ASH      TRTYPE,R0          ; R0 contains amount to point to
                  ; next row
         MOV      CHNSTR,@#UISAR0    ; start at first row
         MOV      #512.,R2
         MOV      R1,-(SP)
         BIS      #1,@#MMSR0
5$:      MOV      TWORDS,R4
10$:     MOV      (R3)+,-(SP)
         MTPI     (R1)+
         SOB      R4,10$
         MOV      (SP),R1
```

```
        ADD     R0,@#UISAR0
        SOB     R2,5$
        TST     (SP)+
        ENDTRN
;
        .SBTTL  FFT FILL
;+
; CALL FFTFIL
; Used to complete the Right half of the centered FFT
;-
FFTFIL:: TST    IFLG
        BNE     12$
        JSR     PC,INIT
12$:    MOV     #1700,@#UISAR0
        MOV     #101500,@#UISAR1
        BIS     #1,@#MMSR0
        MOV     #511.,R5
1$:     MOV     #255.,R4
        MOV     #10,R0
        MOV     #30000,R1
3$:     MOV     #CONBUF,R2
        MOV     #4,R3
5$:     MFPI    (R0)+
        MOV     (SP)+,(R2)+
        SOB     R3,5$
;
; Here we have one complex element in conbuf
;
        MOV     #CBUF2,R2
        NEGF    (R2)+
        MOV     #4,R3
10$:    MOV     -(R2),-(SP)
        MTPI    -(R1)
        SOB     R3,10$
        SOB     R4,3$
        ADD     #100,@#UISAR0
        SUB     #100,@#UISAR1
        SOB     R5,1$
;
; Now get right half of top row from
; bottom half of left column.
;
        MOV     #1600,@#UISAR0
        MOV     #41700,@#UISAR1
        MOV     #4010,R0
        MOV     #255.,R2
20$:    MOV     #20000,R1
        MOV     #4,R3
25$:    MFPI    (R1)+
        MTPI    (R0)+
```

```
        SOB     R3,25$
        ADD     #100,@#UISAR1
        SOB     R2,20$
        ENDTRN
;
        .SBTTL  DATA AND STORAGE
;
CHNSTR: .WORD   11000     ; Start Page address of current channel
TRANSF: .WORD   256.      ; word transfers/line of current channel
TWORDS: .WORD   0         ; word to hold number of words/element
TRTYPE: .WORD   0         ; word to hold current data type
SHIFTS: .WORD   3
XFER:   .WORD   256.      ; word transfers for a line of bytes
BOFF:   .WORD   10000     ; Page address used for 512X512 bytes
NEXAVA: .WORD   1600      ; base Page address - 28kW
CHNO:   .BLKW   30.       ; channel description area 4 channels
CHUSE:  .WORD   0         ; Channel use bit map
NOCHS:  .WORD   0         ; No. of available 256Kb Channels
TEMP:   .WORD   0         ; Temporary storage
IFLG:   .WORD   0         ; Init done flag
MEMFLG: .WORD   0         ; Meminit done flag
CONBUF: .BLKW   2
CBUF2:  .BLKW   2
;
        .END
```

```
        .TITLE EXVAR-Execute Local Variance
        .IDENT   /V01.00/
        .SBTTL   DOCUMENTARY
        .REM     *
This routine calculates the 5X5 local variance image of an image stored
in Frame buffer 0 after creating the Local Mean of that image and
storing it in Frame buffer 1.  The calculations are performed by moving
the image FB "over" the summing FB.  FBSUM is a 16 bit Frame Buffer.
The division by 25 is performed using a product expansion that can be
implemented by multiplys and shifts on the image processor ALU. *

        .SBTTL   OTHER RESOURCES
        .INCLUDE /DK:ITEHED.MAC/   ; Rgister offsets and it definitions

        .MACRO   FBCYCL FBAREG,?$$$
        BIS      #FC.AC1,FB.CTRL(FBAREG)     ; Set single Frame Acquire
$$$:    BIT      #FC.STAT,FB.CTRL(FBAREG)    ; Wait until done
        BNE      $$$
        ADD      #SPINK,FB.PAN(FBAREG)       ; Update Pan register
        .ENDM    FBCYCL

        .MACRO FBCLR ?$$$
        MOV      #0,FB.DATA(R1)              ; Set zero into data
register
        BIS      #FC.AC0,FB.CTRL(R1)         ; Issue clear command
$$$:    BIT      #FC.STAT,FB.CTRL(R1)        ; Wait until done
        BNE      $$$
        .ENDM    FBCLR

        .MACRO   LOAALU CODE
        MOV      R1,-(SP)
        MOV      CODE,R1
        JSR      PC,ALUSET
        MOV      (SP)+,R1
        .ENDM    LOAALU

; Image processor hardware base addresses.
ALUBA=170200
FBSUM=170320       ; Summation FB
FBIM=170000        ; Original Image FB
FBMEAN=170300      ; Location of local mean image
;
EXVAR:: MOV      #ALUBA,R0          ; Get ALU base address
        LOAALU   #MEAN              ; Setup ALU to do MEAN
        MOV      #FBSUM,R1          ; Get FBsum base address.
        FBCLR                       ; Clear Sum Buffer
        MOV      #5,XTEMP           ; Neighborhood X size
        MOV      #5,R3              ; Neighborhood Y size
        MOV      R3,YTEMP
        MOV      #FBIM,R4           ; Base address of FB source Image
```

```
; Do Local Mean!
; R0 = ALU Base Address
; R1 = FB SUM Base Address
; R3 = Y counter
; R4 = FBIM base address
;
        ADD     #2,FB.SCROLL(R4)  ; Offset image from FBSUM
        ADD     #2,FB.PAN(R4)
MYLOOP: MOV     XTEMP,R2          ; Set X counter
MXLOOP: FBCYCL  R1
        DEC     FB.PAN(R4)        ; Slide FBIM left
        SOB     R2,MXLOOP
        ADD     XTEMP,FB.PAN(R4)  ; Remove left slides
        DEC     FB.SCROLL(R4)     ; Slide up
        SOB     R3,MYLOOP
        ADD     YTEMP,FB.SCROLL(R4)       ; Remove up slides
        SUB     #2,FB.SCROLL(R4)  ; Restore FBIM to normal position
        SUB     #2,FB.PAN(R4)
        JSR     PC,DODIV          ; Do a divide by 25 on IP
;
; Now Copy Local mean image to FB 1
        LOAALU  #COPY
        MOV     #FBMEAN,R2        ; Get address of FB to store mean
        FBCYCL  R2
; Do Local Variance
; R0 = ALU Base Address
; R1 = FB SUM Base Address
; R3 = DY counter
; R4 = FB raw base address
        FBCLR
        MOV     YTEMP,R3
        ADD     #2,FB.SCROLL(R4)
        ADD     #2,FB.PAN(R4)
VYLOOP: MOV     XTEMP,R2          ; Set X counter
VXLOOP: LOAALU  #TVAR1
        FBCYCL  R1
        LOAALU  #TVAR2
        FBCYCL  R1
        LOAALU  #TVAR3
        FBCYCL  R1
        LOAALU  #TVAR2
        FBCYCL  R1
        DEC     FB.PAN(R4)        ; Slide output left
        DEC     R2
        BEQ     5$
        JMP     VXLOOP
5$:     ADD     XTEMP,FB.PAN(R4)  ; Remove left slides
        DEC     FB.SCROLL(R4)     ; Slide up
        DEC     R3
        BEQ     6$
        JMP     VYLOOP
```

```
6$:     ADD     YTEMP,FB.SCROLL(R4)     ; Remove up slides
        SUB     #2,FB.SCROLL(R4)
        SUB     #2,FB.PAN(R4)
        JSR     PC,DODIV
        RTS     PC
;
; Now do division
;
DODIV:  LOAALU  #DIV
        FBCYCL  R1
        MOVB    #156.,AL.K3(R0)
        MOVB    #67,AL.SFT(R0)
        MOVB    #47,AL.IN1(R0)
        FBCYCL  R1
        MOVB    #134.,AL.K3(R0)
        FBCYCL  R1
        RTS     PC

; Routine to set ALU Codes.
; R0=Base address of ALU
; R1=Pointer to ALU code table
;
ALUSET: MOVB    (R1)+,AL.K1(R0)
        MOVB    (R1)+,AL.K2(R0)
        MOVB    (R1)+,AL.K3(R0)
        MOVB    (R1)+,AL.CTRL(R0)
        MOVB    (R1)+,AL.SFT(R0)
        MOVB    (R1)+,AL.MUL(R0)
        MOVB    (R1)+,AL.IN1(R0)
        MOVB    (R1)+,AL.IN2(R0)
        MOVB    (R1)+,AL.IN3(R0)
        MOVB    (R1)+,AL.OUT(R0)
        RTS     PC
; Data and storage
XTEMP:  .WORD   0
YTEMP:  .WORD   0
MEAN:   .BYTE   0,0,0,3,0,0,160,62,2,10
COPY:   .BYTE   0,0,0,3,0,0,167,162,1,10
TVAR1:  .BYTE   0,0,0,3,60,020,0,62,2,10
TVAR2:  .BYTE   0,0,0,11,60,20,20,62,2,10
TVAR3:  .BYTE   0,0,0,3,60,20,21,62,2,10
DIV:    .BYTE   0,0,0,5,65,20,167,62,2,10
        .EVEN

        .END
```

```
        .TITLE HSFX Hsitogram Feature extraction software
        .IDENT   /V01.00/
        .SBTTL   DOCUMENTARY
        .REM     *
This module contains routines that interface with the Histogram Feature
Extraction Board.

HSFX Lookup table operations

CAll HFLLUT( table #, array) - Load LUT

CALL HFRLUT( table #, array) - Read LUT

CALL HFLLIN( table #) - Load linear LUT

Histogram operations

CALL HIS( array, sum, Maximum, scale)

CALL HMXMN( imax, imin)

Feature Extraction Operation

i=FSVAL(iVAL) - Return the number of pixels of intensity value ival

i=FGEVAL(iVAL) - Returns the Number of Pixels of intensity value
                           Greater than or equal to ival

i=FLSVAL(ival, ix, iy) Return the number of pixels found and their x,y
                                location.
        *
        .SBTTL   ADDRESSES AND BIT DEFINITIONS
;
        FBCSR=170010                ; HF-512 register assignment
        HFREG=170100
        HFCR = 0            ;Register offset for HF
        HFSR = 2
        FCR = 4
        HFADR = 6
        HFDAT = 10
        HFINC = 1           ;Control bit assignments
        MAPMDE = 2
        ORLSEL = 4
        VINTEN = 10
        AINTEN = 20
        ACTIN = 40
        HF.HAP   = 3400     ;Do histogram on all pixels in both fields
        HF.COBRF = 100
        OBTYPE = 10         ;Status bit assignments
```

```
        OBRF = 20
        RESET = 360
        .SBTTL   ROUTINES
;
HFLLUT:: TST     (R5)+
        MOV     @(R5)+,R2          ; get the LUT to load
        MOV     (R5),R3            ; get address of table array
;
; Now start loading HF register to set up LUT
;
        MOV     #HFREG,R0          ; get base address
        MOV     #45,(R0)+          ; autoinc mem addr by 2, selut     ;
instead of data
        MOV     #140,(R0)+         ; force set of OBRF to load LUTs
        TST     (R0)+              ; skip feature count register
        SWAB    R2                 ; convert LUT # to address
        ASL     R2
        MOV     R2,(R0)+           ; select LUT
        MOV     #256.,r4           ; the table is 256 words long
10$:    MOVB    (R3)+,(R0)         ; load lut
        SOB     R4,10$
        RTS     PC
;
HFRLUT:: TST     (R5)+
        MOV     @(R5)+,R2          ; get the LUT to read
        MOV     (R5),R3            ; get address of table array
;
; Now start loading HF register to set up LUT
;
        MOV     #HFREG,R0          ; get base address
        MOV     #45,(R0)+          ; autoinc mem addr by 2, selut     ;
instead of data
        MOV     #140,(R0)+         ; force set of OBRF to load LUTs
        TST     (R0)+              ; skip feature count register
        SWAB    R2                 ; convert LUT # to address
        ASL     R2
        MOV     R2,(R0)+ ; select LUT
        MOV     #256.,r4 ; the table is 256 words long
10$:    MOVB    (R0),(R3)+         ; load lut
        SOB     R4,10$
        RTS     PC
;
HFLLIN:: TST     (R5)+
        MOV     @(R5)+,R2          ; get the LUT to load
;
; Now start loading HF register to set up LUT
;
        MOV     #HFREG,R0          ; get base address
        MOV     #45,(R0)+          ; autoinc mem addr by 2, selut
                                   ; instead of data
        MOV     #140,(R0)+         ; force set of OBRF to load LUTs
```

```
        TST     (R0)+           ; skip feature count register
        SWAB    R2              ; convert LUT # to address
        ASL     R2
        MOV     R2,(R0)+        ; select LUT
        MOV     #256.,r4        ; the table is 256 words long
        CLR     R1
10$:    MOV     R1,(R0)         ; load lut
        INC     R1
        SOB     R4,10$
        RTS     PC
;
FSVAL:: TST     (R5)+
        MOV     @(R5)+,R2       ; get the value to compare
;
; Now start loading HF register to set up FEX LUT
;
        MOV     #HFREG,R0       ; get base address
        MOV     #45,(R0)+       ; autoinc mem addr by 2, selut
        ;instead of data
        MOV     #140,(R0)+      ; force set of OBRF to load LUTs
        TST     (R0)+           ; skip feature count register
        MOV     #0,(R0)+ ; will be using FEX LUT 1
        MOV     #256.,r4 ; the table is 256 words long
        TST     R2
        BEQ     15$
10$:    MOV     #0,(R0)         ; load lut up to values as non-
        ;features, LUT is zero
        DEC     R4
        SOB     R2,10$
15$:    MOV     #1,(R0)         ; load the single value feature
        DEC     R4
        TST     R4
        BEQ     30$
20$:    MOV     #0,(R0)
        SOB     R4,20$
30$:    JSR     PC,DOFEC
        RTS     PC
;
FGEVAL:: TST    (R5)+
        MOV     @(R5)+,R2       ; get the value to compare
;
; Now start loading HF register to set up FEX LUT
;
        MOV     #HFREG,R0       ; get base address
        MOV     #45,(R0)+       ; autoinc mem addr by 2, selut
                                ; instead of data
        MOV     #140,(R0)+      ; force set of OBRF to load LUTs
        TST     (R0)+           ; skip feature count register
        MOV     #0,(R0)+        ; will be using FEX LUT 0
        MOV     #256.,r4        ; the table is 256 words long
        TST     R2
```

```
        BEQ     15$
10$:    MOV     #0,(R0)         ; load lut up to values as non-    ;
features, LUT is zero
        DEC     R4
        SOB     R2,10$
15$:    MOV     #1,(R0)         ; load the remaining table with
        ; features
        SOB     R4,15$
        JSR     PC,DOFEC
        RTS     PC
;
FLSVAL:: MOV    @4(R5),R2       ; get the value to compare
;
; Now start loading HF register to set up FEX LUT
;
        MOV     #HFREG,R0       ; get base address
        MOV     #45,(R0)+       ; autoinc mem addr by 2, selut     ;
instead of data
        MOV     #140,(R0)+      ; force set of OBRF to load LUTs
        TST     (R0)+           ; skip feature count register
        MOV     #0,(R0)+        ; will be using FEX LUT 1
        MOV     #256.,r4        ; the table is 256 words long
        TST     R2
        BEQ     15$
10$:    MOV     #0,(R0)         ; load lut up to values as non-    ;
features, LUT is zero
        DEC     R4
        SOB     R2,10$
15$:    MOV     #1,(R0)         ; load the single value feature
        DEC     R4
        TST     R4
        BEQ     30$
20$:    MOV     #0,(R0)
        SOB     R4,20$
30$:    MOV     #1000,R1
        CMP     #2,@2(R5)
        BEQ     32$
        TST     @2(R5)
        BEQ     29$
        BIC     #400,DECID
        BR      31$
29$:    BIS     #400,DECID
31$:    MOV     #0,R1
32$:    JSR     PC,DOFEC
;
;       R0 has the nuber of features found
;       Store values in X,Y arrays
;
        TST     R0
        BLE     50$
        MOV     R0,R2
```

```
        MOV     6(R5),R3 ; X base address
        MOV     10(R5),R4          ; Y Address
        MOV     #HFREG,R1
        MOV     #41,(R1)+          ;Addres Outbuffer with autoinc.
        TST     (R1)+
        TST     (R1)+
        MOV     #0,(R1)+ ;STARTING ADDRESS
40$:    MOV     (R1),(R3)+         ; mov x value
        MOV     (R1),(R4)+         ; mov y value
        SOB     R2,40$
50$:    RTS     PC
;
; Now we want to count the features by setting up the next acquisition ;
mode  in the HF and starting the operation.
;
DOFEC:  MOV     #HFREG,R0
        MOV     R1,(R0)+ ; Do interlaced FEX operation
                                   ;
        MOV     #FBCSR,R2          ; Wait for three fields to pass
        MOV     #2,R4
REDO:   MOV     #3,R3
FWAIT:  BIT     #40,(R0)
DECID:  BEQ     FWAIT
        SOB     R3,FWAIT
;
; Look for 0 to 1 transition of vertical blank
;
102$:   BIT     #40,(R2)
        BNE     102$
103$:   BIT     #40,(R2)
        BEQ     103$
        MOV     #100,(R0)          ; Clear OBRF, start conversion.
32$:    BIT     #20,(R0) ; wait till OBRF is set
        BEQ     32$
        BIT     #10,@#HFREG        ; Check to see if fex data
        BNE     REDO               ; If not do it again
        SOB     R4,REDO
        TST     (R0)+
        MOV     (R0),R1
        MOV     R1,R0    ;Return number of features found in R0
        RTS     PC
;
FEXT::
;       first load lut table.
;
        TST     (R5)+
        MOV     #HFREG,R0
        MOV     #45,(R0)+
        MOV     #140,(R0)+
        TST     (R0)+
        MOV     #0,(R0)+           ;Select FEXlut #0
```

```
        MOV     #256.,R4
        MOV     (R5)+,R3
10$:    MOV     (R3),(R0)
        MOV     (R3)+,R2
        SOB     R4,10$
        MOV     #HFREG,R0
        MOV     #41,(R0)+       ;Set output buffer to auto
                                ;increment.
        MOV     #100,(R0)       ;Clear OBRF, start conversion.
20$:    BIT     #20,(R0)
        BEQ     20$
        TST     (R0)+
        MOV     (R0),@(R5)+     ;Return number of features found.
        MOV     (R0)+,R2        ;Get number of features
        ASR     R2              ;Double it.
        RTS     PC
;
GETDAT:: TST    (R5)
        MOV     (R5),R3
        MOV     #<HFADR!HFREG>,R0
        MOV     #0,(R0)+ ;STARTING ADDRESS
30$:    MOV     (R0),(R3)+
        SOB     R2,30$
        RTS     PC
;
HIS::   CLR     R1
        JSR     PC,DOHIS
        RTS     PC
;
RSAVE:  .WORD   0
HISRT:: MOV     #10000,R1
        JSR     PC,DOHIS
        RTS     PC
;
; Do histogram operation.
; R1 contains the Histogram operation to be performed.
; What is to be loaded into HF-512 CSR
;
DOHIS:: MOV     #HFREG,R0
        BIS     #HF.HAP,R1
        MOV     R1,(R0)+
        MOV     #HF.COBRF,(R0)  ;Clear OBRF, to transfer last oper
                                ;data
20$:    BIT     #20,(R0)
        BEQ     20$
        MOV     #100,(R0)       ;Clear OBRF, start conversion.
21$:    BIT     #20,(R0)
        BEQ     21$
        MOV     #100,(R0)       ;Clear OBRF, start conversion.
24$:    BIT     #20,(R0)
        BEQ     24$
```

```
        MOV     #100,(R0)
22$:    BIT     #20,(R0)
        BEQ     22$
        MOV     #100,(R0)
23$:    BIT     #20,(R0)
        BEQ     23$
;
;       Store values in histogram array as reals
;
        MOV     2(R5),R3
        MOV     R3,R4
        MOV     #HFREG,R0
        MOV     #41,(R0)+           ;Addres Outbuffer with autoinc.
        TST     (R0)+
        TST     (R0)+
        CLRF    SUM
        LDF     SUM,1
        LDF     SUM,2
        MOV     #256.,R2
        MOV     #0,(R0)+ ;STARTING ADDRESS
        SETL                        ; set long integer mode for
        ;conversion.
30$:    MOV     (R0),2(R3)
        MOV     (R0),(R3)+
        TST     (R3)+
        LDCLF   (R4),0
        STF     0,(R4)
        ADDF    (R4),1
        CMPF    (R4),2
        CFCC
        BLT     35$
        LDF     (R4),2
35$:    ADD     #4,R4
        SOB     R2,30$
        STF     1,@4(R5)
        STF     2,@6(R5)
        LDF     SCALE,2
        DIVF    @6(R5),2
        STF     2,@10(R5)
        SETI
        RTS     PC
;
SUM:    .BLKW   2
SCALE:  .FLT2   480.
;
FEX::   TST     (R5)+
        MOV     #HFREG,R0
        MOV     #45,(R0)+
        MOV     #140,(R0)+
        TST     (R0)+
        MOV     #0,(R0)+            ;Select FEXlut #0
```

```
        MOV     #256.,R4
        MOV     (R5)+,R3
10$:    MOV     (R3),(R0)
        MOV     (R3)+,R2
        SOB     R4,10$
        MOV     #HFREG,R0
        MOV     #1000,(R0)+
        MOV     #100,(R0)           ;Clear OBRF, start conversion.
20$:    BIT     #20,(R0)
        BEQ     20$
        MOV     #100,(R0)
22$:    BIT     #20,(R0)
        BEQ     22$
        MOV     #100,(R0)
23$:    BIT     #20,(R0)
        BEQ     23$
        TST     (R0)+
        MOV     (R0),@(R5)+         ;Return number of features found.
        MOV     (R5),R3
        MOV     #HFREG,R0
        MOV     #41,(R0)+           ;Addres Outbuffer with autoinc.
        TST     (R0)+
        MOV     (R0)+,R2
        TST     R2
        BEQ     40$
        ASL     R2
        MOV     #0,(R0)+ ;STARTING ADDRESS
30$:    MOV     (R0),(R3)+
        SOB     R2,30$
40$:    RTS     PC
;
HMXMN:: MOV     #HFREG,R0           ; get base address
        MOV     #45,(R0)+           ; autoinc mem addr by 2, selut      ;
                                    ; instead of data
        MOV     #140,(R0)+          ; force set of OBRF to load LUTs
        TST     (R0)+               ; skip feature count register
        MOV     #2000,(R0)+         ; select HIS LUT 0
;
; Load linear table 0 to 255
;
        MOV     #256.,r4 ; the table is 256 words long
        CLR     R1
10$:    MOV     R1,(R0)             ; load lut
        INC     R1
        SOB     R4,10$
;
; Perform histogram operation
;
        MOV     #HFREG,R0
        MOV     #3400,(R0)+
        MOV     #100,(R0)           ;Clear OBRF, start conversion.
```

```
20$:    BIT     #20,(R0)
        BEQ     20$
        MOV     #100,(R0)       ;Clear OBRF, start conversion.
21$:    BIT     #20,(R0)
        BEQ     21$
        MOV     #100,(R0)       ;Clear OBRF, start conversion.
24$:    BIT     #20,(R0)
        BEQ     24$
        MOV     #100,(R0)
22$:    BIT     #20,(R0)
        BEQ     22$
        MOV     #100,(R0)
23$:    BIT     #20,(R0)
        BEQ     23$
;
; Check values to find max
;
        MOV     #HFREG,R0
        MOV     #41,(R0)+       ;Addres Outbuffer with autoinc.
        TST     (R0)+
        TST     (R0)+
        MOV     #256.,R2
        MOV     #0,(R0)+ ;STARTING ADDRESS
        MOV     #HREGS,R3
30$:    MOV     (R0),(R3)       ; get low word
        MOV     (R0),R1
        BIS     R1,(R3)+        ; get high word
        SOB     R2,30$
;
; Now check hregs for min and max
;
        MOV     #256.,R1
40$:    TST     -(R3)
        BNE     50$
        SOB     R1,40$
50$:    MOV     R1,@2(R5)
        MOV     #256.,R0
        MOV     #1,R1
        MOV     #HREGS,R3
60$:    TST     (R3)+
        BNE     70$
        INC     R1
        SOB     R0,60$
70$:    MOV     R1,@4(R5)
        RTS     PC
;
HREGS:  .BLKW   256.
;
        .END
```

```
        .TITLE IPFIO-Image Processor File Input/Output Software
;
        .IDENT  /V01.00/
;
        .SBTTL  DOCUMENTARY

        .REM    *
```

This software contains two subroutines that perform Frame Buffer I/O operations to and from disk files. The files created are 512X512X8 images of the byte-wide frame buffers of the ITI IP-512 image processor sub-system. The file format is based on a line of data for each disk block giving a total file length of 512 blocks. The two functions contained in this module are WriTe-FILE (WTFILE) and ReaD-FILE (RDFILE). The form of the calls are given below:

```
CALL WTFILE(I_FB,I_Available_Channel,RAD50_Filename,I_error_Flag)

CALL RDFILE(I_FB,I_Available_Channel,RAD50_Filename,I_error_Flag)
```

where,

I_FB = 0,1,2	The frame buffer to be written to or read from.
I_Available_Channel	An available RT-11 I/O channel.
RAD50_Filename	4 Word RAD50 representation of the filename of the data to be accessed.
I_error_flag = -1	Indicating an error back to the calling program if the data transfer was not performed successfully.

```
                *
        .SBTTL  RESOURCES
;+
;  We need to include the FB definitions and header file
;  for proper assembly of this module.
;-
;
        .INCLUDE        /DK:ITEHED.MAC/
;+
;  System Programmed Requests used by this program
;-
;
        .MCALL  .PRINT,.READW,.WRITW,.ENTER
        .MCALL  .LOOKUP,.CLOSE,.FETCH
;+
; Some useful multiple bit definitions
;-

FC.IXARW=FC.SAR!FC.SAW!FC.AXD!FC.EXC      ; Auto inc x after read or
                        ; write
```

```
        .SBTTL  WRITE FILE FROM FRAME BUFFER
;CALL WTFILE(IFB,I_Available_Channel,RAD50_Filename,I_error_Flag)
WTFILE::
        FBDBA   2,R1
        MOV     @R1,R1              ; Get base address of frame buffer
        MOV     @4(R5),CHANL        ; Get the channel number
        MOV     6(R5),FILE          ; Pointer to file name
        MOV     10(R5),ERRPTR       ; Pointer to error word
        CLR     @ERRPTR             ; No errors yet.
        .FETCH  LIMIT,FILE          ; Fetch device handler into memory if
                                    ; needed
        BCC     1$                  ; If successful go
        MOV     #FETERR,R1          ; Else show error
        JMP     RWERR               ; Process error
1$:     .LOOKUP #AREA,CHANL,FILE ; See if the file already exists
        BCS     30$                 ; If not, go to create
        MOV     #NAMERR,R1          ; Tell caller file exists
        JMP     RWERR               ; Process error
30$:    .ENTER  #AREA,CHANL,FILE,#-1     ; Open a file.
        BCC     5$                  ; Go write if open was  successful
        MOV     #OPERR,R1           ; Show error on open
        JMP     RWERR               ; Process error
5$:     CLR     R5                  ; Reset block pointer
        BIC     #FC.EYC,FB.CTRL(R1)       ; Make sure auto y inc is
                ; disabled
        BIS     #FC.IXARW,FB.CTRL(R1)     ; Auto inc x on read or write
        CLR     FB.Y(R1)                  ; Reset current video line
                        ; counter
        MOV     FB.PAN(R1),FB.X(R1)       ; Adjust veiw on screen to x
                                          ; position
        MOV     #128.,R4
70$:    MOV     #4,R3
        MOV     #OUTBUF,R0                ; Get address of output buffer
6$:     MOV     #512.,R2                  ; 512 pixels per line
60$:    MOVB    FB.DATA(R1),(R0)+         ; Put data in buffer
        SOB     R2,60$                    ; See if done with the line
        INC     FB.Y(R1)                  ; Next line
        SOB     R3,6$                     ; See if ready to write
;+
;   Write 4 video lines of data
;-
        .WRITW  #AREA,CHANL,#OUTBUF,#1024.,R5
        BCC     25$                       ; Go if no error
        MOV     #WERR,R1                  ; show error
        JMP     RWERR                     ; Process error
25$:    ADD     #4,R5                     ; Bump block pointer by 4
        SOB     R4,70$                    ; Check if done with frame
        JMP     RWDONE
```

```
        .SBTTL  COMMON ERROR AND DONE PROCESSING
;+
;  Common error processing
;-
RWERR:  MOV     #1,@ERRPTR
        .PRINT  R1
        .PRINT  #CRLF
;+
;  Common Transfer complete processing
;-
RWDONE: .CLOSE  CHANL           ; Close used channel
        RTS     PC              ; Return to caller
;
        .SBTTL  READ FILE INTO FRAME BUFFER
;CALL RDFILE(IFB,I_Available_Channel,RAD50_Filename,I_error_Flag)
RDFILE::
        MOV     @2(R5),R3       ; Get frame buffer number
        FBDBA   2,R1
        MOV     @R1,R1          ; Get base address of frame buffer
        MOV     @4(R5),CHANL    ; Get the channel number
        MOV     6(R5),FILE      ; Pointer to file name
        MOV     10(R5),ERRPTR   ; Get address of error word
        CLR     @ERRPTR         ; No errors yet.

        .FETCH  LIMIT,FILE      ; Fetch device handler if necessary
        BCC     1$              ; go if successful
        MOV     #FETERR,R1      ; Show error
        JMP     RWERR           ; Process error

1$:     .LOOKUP #AREA,CHANL,FILE ; See if file exists
        BCC     5$                      ; Go if it is
        MOV     #FILERR,R1
        JMP     RWERR
;+
;  View frame buffer that we are going to restore
;-
5$:     MOV     ALBADR,R5
        MOVB    R3,AL.IN3(R5)
;+
;  Clear Frame buffer that we are going to write to
;-
        CLR     FB.DATA(R1)
        BIC     #FC.STAT,FB.CTRL(R1)    ; Force stop of current
                ; operation
        BIS     #FC.ACO,FB.CTRL(R1)     ; Set all pixs to fb.data
2$:     BIT     #FC.STAT,FB.CTRL(R1)    ; Wait till done
        BNE     2$
;+
;  Set up for transfers
;-
        CLR     FB.PAN(R1)              ; Reset view on screen
```

```
        CLR     FB.SCROLL(R1)
        CLR     FB.Y(R1)                ; Start at 0,0
        CLR     FB.X(R1)
        CLR     R5                      ; Reset block pointer
        BIC     #FC.EYC,FB.CTRL(R1)     ; Ensure Auto Y inc is off
        BIS     #FC.IXARW,FB.CTRL(R1)   ; Auto inc x on read
;+
;  Read 4 video lines into buffer
;-
        MOV     #128.,R4
70$:    .READW  #AREA,CHANL,#OUTBUF,#1024.,R5
        BCC     25$                     ; Go if Read was successful
        MOV     #RERR,R1                ; Else show error
        JMP     RWERR                   ; process error
25$:    MOV     #4,R3
        MOV     #OUTBUF,R0
26$:    MOV     #512.,R2
60$:    MOVB    (R0)+,FB.DATA(R1)       ; Unload bueffer into register
        SOB     R2,60$                  ; Check if done with line
        INC     FB.Y(R1)                ; Next line
        SOB     R3,26$                  ; See if done with buffer
        ADD     #4,R5           ; Bump block pointer
        SOB     R4,70$          ; Is transfer complete
        JMP     RWDONE
;
        .SBTTL  STORAGE AND DATA
;+
;  General storage
;-
AREA:   .BLKW   10.     ; Emt area for Programmed Requests
OUTBUF: .BLKW   1024.   ; I/O buffer
CHANL:  .WORD   0       ; Address of I/O channel
FILE:   .WORD   0       ; Address of File name
ERRPTR: .WORD   0       ; Address of error return word
;+
;  Error messages
;-
CRLF:   .ASCIZ  <12><15>
RERR:   .ASCIZ  /? FILE READ ERROR ?/
WERR:   .ASCIZ  /? FILE WRITE ERROR ?/
OPERR:  .ASCIZ  /? FILE OPEN FAILURE ?/
FILERR: .ASCIZ  /? FILE NOT FOUND ?/
NAMERR: .ASCIZ  /? FILE ALREADY EXISTS ?/
FETERR: .ASCIZ  /? DEVICE FETCH FAILED ?/
        .EVEN
        .LIMIT          ; Job address limit at link time for device
                        ; driver fetches
LIMIT = .-2
;
        .END
```

```
C
      PROGRAM POLY
C
C  This program performs a 2nd-order polynomial fit on an image in FB0,
C puts the fitted image temporarily in FB1, and subtracts the two and
C puts the result in FB1.
C
      INTEGER  TEMPL(14,12),I(512),SUMBLK(31),EVABLK(47)
      REAL     R(512),X(512),X2(512)
C
C Define array processor command structures.
C
C                                                    !Opcode  function
      DATA     SUMBLK/"006100,"170016,"377,
      1              "177776,                      !014 Transfer data
      2       "016100,0,0,0,0,0,"177774,           !034 Sum and store
      3       "111100,0,0,0,0,0,"177774,0,0,0,     !222 Mult and Sum
      4       "111100,0,0,0,0,0,"177774,0,0,0/     !222 Mult and Sum
C
      DATA     EVABLK/"044100,0,0,0,0,0,0,0,0,
      1              "177774,                      !110 Scal Mult.
      2       "046100,0,0,0,0,0,0,0,0,"177774,     !114 Scal Mult.
      3       "032100,0,0,0,0,0,0,0,0,0,           !064 ACC+Vect
      4       "036100,0,0,0,0,0,0,0,0,"177774,     !074 ACC+scal
      5       "012100,0,0,0,"170316,"377,"177776/ !024 Transfer data
C
      DATA     XFACT/32768.0/              ! Array processor conversion.
C
C Constants for Gausian Ellimination for a 512 pixel line.
C
      DATA     A02/87125.5/
      DATA     A01/255.5/
      DATA     A00/1.0/
      DATA     A11/-42.62439/
      DATA     A10/-.501466/
      DATA     A20/.1008829/
      DATA     F0/1.501466/
      DATA     F1/1.801755/
      DATA     F2/1.799998/
      DATA     SUM/512./
      DATA     SUMX/130816.0/
      DATA     SUMX2/44608.28E3/
C
      F3=F2*F0-F1
C
C Set up image processor
C
      CALL SELGRP(1)              !Setup software pointers
      CALL FBSET(0,1)             !FB0+FB1 set for AP access
```

```
      CALL SELVUE(1)              !View FB1, Fit
      ISPN=(LKPAN(1).AND."777)    !Make sure pan values are right
      CALL SETSPN(1,ISPN)
      ISPN=(LKPAN(0).AND."777)
      CALL SETSPN(0,ISPN)
      CALL SETADM(1)
C
      CALL VINIT(TEMPL,12)        !Set up area for AP command structures
      CALL VRAMP(0.0,1.0,X,1,512)          !Make a linear ramp, 0-511
C
C Get memory starting address of all elements and arrays that will be
C accessed by the AP.
C
      CALL VADDR(IX2ADR,X2)
      CALL VADDR(IXADR,X)
      CALL VADDR(IRADR,R)
      CALL VADDR(ISFADR,SUM1)
      CALL VADDR(ISFXAD,SUM2)
      CALL VADDR(ISFXXA,SUM3)
      CALL VADDR(IA0ADR,A0)
      CALL VADDR(IA1ADR,A1)
      CALL VADDR(IA2ADR,A2)
C
C Establish addresses in command structures that will not  change
C
      SUMBLK(9)=ISFADR
      SUMBLK(16)=ISFXAD
      SUMBLK(26)=ISFXXA
C
      EVABLK(8)=IA1ADR
      EVABLK(18)=IA2ADR
      EVABLK(38)=IA0ADR
C
      CALL VWAIT
      CALL VMUL(X,1,X,1,X2,1,512,'R')      !Get ramp squared
      CALL VWAIT
C
      DO 200 J=1,480
      IY=J-1
C
      SUMF=0.
      SUMFX=0.
      SUMFX2=0.
C
C  Set up address that will need to be bumped
C
      SUMBLK(19)=IXADR
      SUMBLK(29)=IX2ADR
C
      DO 20 M=1,8
C
```

```
        CALL VBLK(SUMBLK,4)
        CALL VBLK(SUMBLK(5),7)
        CALL VBLK(SUMBLK(12),10)
        CALL VBLK(SUMBLK(22),10)
C
        SUMBLK(19)=SUMBLK(19)+"400
        SUMBLK(29)=SUMBLK(29)+"400
C
        CALL VWAIT
C
        SUMF=SUMF+SUM1
        SUMFX=SUMFX+SUM2
        SUMFX2=SUMFX2+SUM3
C
20      CONTINUE
C
        FX2OX2=XFACT*SUMFX2/SUMX2
        FXOX=XFACT*SUMFX/SUMX
        FO=XFACT*SUMF/SUM
C
        AO=(FX2OX2-F2*FXOX+F3*FO)/A20
        A1=(FXOX-FO*FO-A10*AO)/A11
        A2=(FO-A01*A1-A00*AO)/A02
C
        AO=AO/XFACT
        A1=A1/XFACT
        A2=A2/XFACT
C
C Now evaluate for each x in the line
C
        EVABLK(2)=IXADR
        EVABLK(5)=IRADR
        EVABLK(12)=IX2ADR
        EVABLK(28)=IRADR
C
        DO 150 L=1,8
C
        CALL VBLK(EVABLK,10)
        CALL VBLK(EVABLK(11),10)
        CALL VBLK(EVABLK(21),10)
        CALL VBLK(EVABLK(31),10)
        CALL VBLK(EVABLK(41),7)
C
        EVABLK(2)=EVABLK(2)+"400
        EVABLK(5)=EVABLK(5)+"400
        EVABLK(12)=EVABLK(12)+"400
        EVABLK(28)=EVABLK(5)
        CALL VWAIT
150     CONTINUE
        CALL FBINC                  !Bump FBs for access to next line.
200     CONTINUE
```

```
      CALL VDONE                    !Let AP know that we are done
C
C Setup IP to do the subtraction.
C
      CALL SETK(1,0)                !Constants to Zero
      CALL SETK(2,0)
      CALL SETK(3,0)
      CALL SETOPC(2,1)
      CALL SUALFN(0,4,1,4,0,2,0,0,0,0,1,3)
      CALL DOALFN(1,0,0,1,0)        !Do subtraction
C
C Set up IP to take the absolute value.
C
      CALL SETOPC(4,0)
      CALL SUALFN(2,4,3,4,0,4,0,0,0,0,0,1)
      CALL DOALFN(1,0,1,0,0)        !Do Absolute value
C
C Now scale the data to full 8 bits
C
      CALL HMXMN(IMAX,IMIN)         !Get limits of image data
C
C Subtract minimum
C
      CALL SETOPC(1,1)
      CALL SETK(3,imin-1)
      CALL SUALFN(4,4,1,4,0,1,0,0,0,0,0,1)
      CALL DOALFN(1,0,1,0,0)        !Do subtraction
C
C Multiply to 8 bits
C
      IFACT=255/(IMAX-IMIN)-1
      CALL SETOPC(3,0)  !A+B
      CALL SETK(3,IFACT)
      CALL SUALFN(4,1,1,4,1,3,0,0,0,0,0,1)
      CALL DOALFN(1,0,1,0,0)        !Do multiply
C
      STOP
      END
```

VITA

James Russell Younkin was born on March 28, 1957 in Maryville, Tennessee. He attended Panorama Elementary and Shugart Junior High Schools in Hillcrest Heights, Maryland. He returned to Tennessee where he graduated from Oak Ridge High School in June 1975. The following September he entered the University of Tennessee, Knoxville and in March 1980 he received a Bachelor of Science degree in Electrical Engineering basing his senior year of study in analog electronics. Immediately after attaining this degree, he began study toward a Master of Science degree in Electrical Engineering. After two quarters of full time study in the graduate program, he accepted a Development Division position at the Y-12 Plant in Oak Ridge and has continued his pursuit of the advanced degree through the Oak Ridge study program in the evenings. His duties at Y-12 include integration of electronic instrumentation and computer hardware and software development for small automated data acquisition and process control systems. The degree is to be completed June 1986 with his two areas of interest being logic design and image processing.