1988

Development and implementation of a hardware method for edge detection in computer images

Dimitrios Koutsounis
Iowa State University

Order Number 9003543

Development and implementation of a hardware method for edge detection in computer images

Koutsounis, Dimitrios, Ph.D.

Iowa State University, 1988

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

Development and implementation of a hardware method for edge detection in computer images

by

Dimitrios Koutsounis

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Department: Electrical Engineering and Computer Engineering
Major: Computer Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1988

TABLE OF CONTENTS

I. INTRODUCTION

Computer-aided image analysis has advanced in lock-step with the development of more and more powerful computational engines and graphics displays, combined with increasingly sophisticated software to implement digital filtering and related analysis tools. Still, there remains a strong demand for improved systems capable of extracting features and clarifying complex pictorial images such as those obtained in medical imaging or satellite surveillance, and for performing this type of analysis in an interactive manner. Ideally, an operator could manipulate a raw image, perhaps one generated in real time, in an intuitive manner without predetermination of the exact filtering algorithm or analysis technique. This would permit the full range of available tools to be invoked by the operator, who could optimize the choice of tools by direct observation on the effects on the image under study.

Among the more important classes of image features to be extracted are the so-called "edges" which define boundaries between the principal areas of the image under study. A variety of software-based schemes for identifying and displaying such features are available but, unless very high-performance computing is used, the extraction of edge information is far too slow to permit the desired interactive manipulation.

This dissertation will describe a technique which has been proven to provide the desired high-speed interactive capability for manipulation of edge imagery. Since the technique is implemented through modification of circuit components rather than through the more customary software-based methods, it will be referred to as the hardware edge detection method. Although it is intrinsically quite simple and is largely based on empirical experimentation, its performance is quite remarkable. Since the technique has meaning only within the context of performance of an interactive image manipulation system, a rather complete description of such a system is also provided. This system has been developed by the author to exploit the potential of the basic technique. This dissertation will provide an extensive view of experimental results obtained with the system.

To provide a firm basis for comparison of the performance of the described method to existing methods, several well-known algorithms for edge detection have been implemented and applied to the same images explored with the new hardware edge detection method. The required processing times for the software, as implemented on an IBM PC-class system, are shown to be orders of magnitude too long if interactivity is to be achieved.

After development of this set of baseline values, this

dissertation will establish a foundation for understanding of the edge detection technique by providing a fairly detailed description of the image processing system in which the technique is to be implemented. Following this, the basic technique will be described and experimental results obtained on a wide variety of images will be given.

II. EDGE DETECTION TECHNIQUES: AN OVERVIEW

Isolated points and thin lines are not frequent occurrences in most applications of practical interest. Edge detection is by far the most common approach for detecting meaningful discontinuities at the gray level.

An edge can be defined as the boundary between two regions with relatively distinct gray level properties [1]. Edge detection, on the other hand, is the process that locates a boundary between two adjacent regions of an image which differ in one or more characteristics. In this case, the gray levels are the critical characteristics.

Boundaries of objects in an image tend to show up as intensity discontinuities. Experiments with the human visual system show that boundaries in images are extremely important features; often an object can be recognized from only a crude outline [2]. This fact provides the principal motivation for representing objects by their boundaries.

Edge detection is a very important aspect of feature extraction in image processing applications. Much research has been done on this subject [3-7]. The list for the applications that use or need edge detection methods is very long [8]. We will present some areas where these methods are of paramount importance.

In the general application domain, edge detection is used in images for location of objects, dimension

measurements, area measurements, motion measurements, data compression, and other purposes.

In the robotics domain, edge detection is used in outdoor and indoor scenes for recognition and description of objects, detection and identification of product components for automated assembly lines, non-contact measurements, detection of faults in production, and other purposes.

In the medical field, edge detection is used in Computer-Aided Tomography (CAT) and nuclear medicine for identification of cells and chromosomes, measurements of nuclear medicine images for diagnosis, and other purposes.

In the satellite and aerial domain, edge detection is used for terrains, runaways, buildings, and meteorological images, geographic mappings, target locations, and other purposes.

In the military domain, edge detection is of paramount importance for aerial images, including vehicle detection and classification, satellite images, scene matching for missile guidance and target acquisition, spying and tactical analysis.

Most edge detection techniques are based on the computation of local derivative operators [1]. This concept can be illustrated with the aid of Figure 1(a) which shows an image of a white simple object on a dark background.

It also shows the gray-level profile along a horizontal

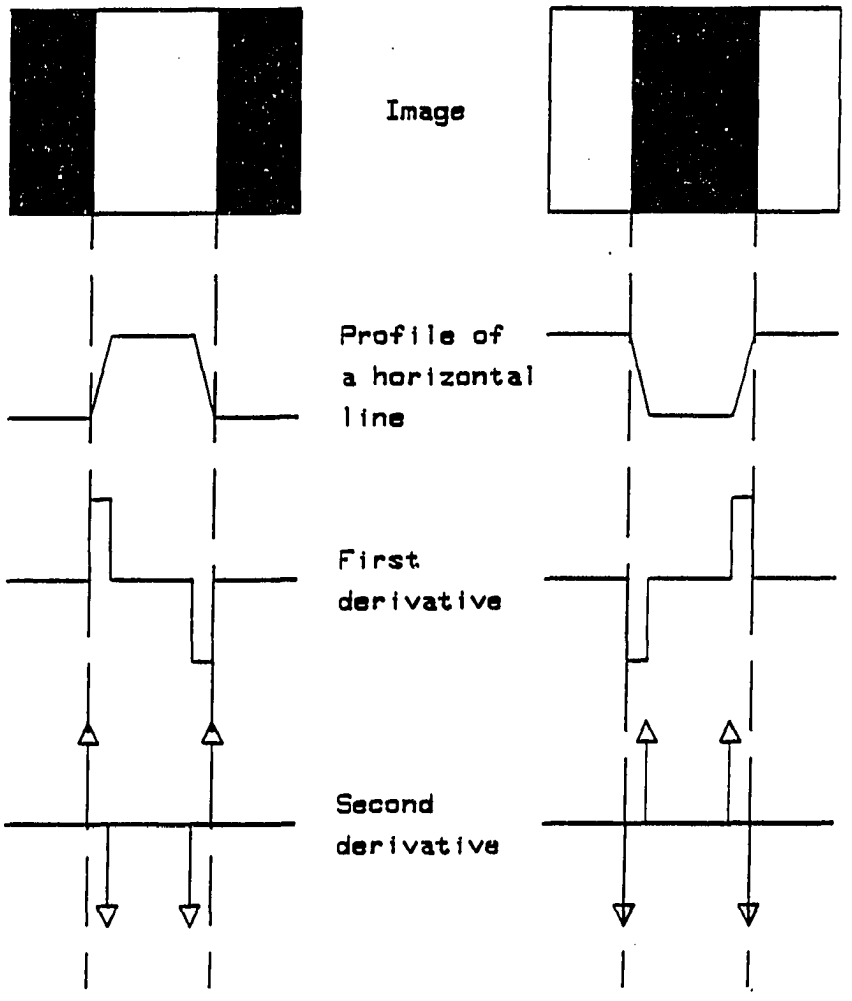


Figure 1. Edge detection operators

scan line of the image, and the first and the second derivatives of the profile. It should be noted from the profile that the edge (actually the transition from dark to light) is modeled as a ramp, because edges in digital images are generally slightly blurred.

The first derivative of an edge modeled in this manner is 0 in all regions of constant gray-level, and it assumes a

constant value during a gray-level transition. The second derivative, on the other hand, is 0 in all locations, except at the onset and termination of a gray-level transition.

The magnitude of the first derivative can be used to detect the presence of an edge, while the sign of the second derivative can be used to determine whether an edge pixel lies on the dark (background) or light (object) side of an edge. The sign of the second derivative in Figure 1(a) is positive for pixels lying on the dark side of these edges and negative for pixels lying on the light side of the edges. The same applies for the case of a dark object on a light background, Figure 1(b).

The general techniques of edge detection are heuristic in nature. Each method must be "tuned" by trial-and-error procedures using a set of test pictures.

Usually the resulting performance can only be measured, not predicted. Most edge detection methods fail to cope effectively with noise. Their performance in the presence of noise can be improved by statistically-based processing techniques [3].

Some factors for developing performance criteria for an edge detector are [8]:

- to determine the pixel location of an edge,
- to include height and slope angle of an edge, and its spatial orientation, and

- to have a confidence factor associated with the edge decision (the closeness of fit between actual image data and the idealized edge model).

A performance evaluation may be difficult because of:

- the lack of definitive performance criteria,
- the large number of edge detection methods, and
- the difficulties in determining the best parameters for each method.

Relatively few studies of edge detector performance have been reported in the literature [9,10].

A common strategy in edge detection is to establish some bound on the probability of false detection resulting from noise and then to maximize the probability of true signal detection.

This concept involves the setting of the edge detection threshold at a level such that the probability of false detection resulting from noise alone does not exceed some desired value. The probability of true edge detection can be evaluated by a coincidence comparison of the edge maps of an ideal and an actual edge detector.

There are three major types of errors associated with determination of an edge location [3]:

1. failure to detect valid edge points,
2. failure to localize edge points, and
3. classification of noise pulses as edge points.

III. CONVENTIONAL SOFTWARE EDGE DETECTION METHODS

Some classical software edge detection methods that were used in processing the image of Photograph 1 will be presented in this section. This image is taken by the frame grabber, which will be described in Chapter IV. The original picture, without processing, from the title of the book [11] is shown in Photograph 1.

The unique characteristic of this image is that it has only two gray-level values, corresponding to black letters on a white background converted to an inverse format. It is clear that the photograph does not have a "good" black and white image due to a number of factors, e.g., the use of a color video camera, the non-uniformity of space and surface lighting, the noise introduced by the lenses and the frame grabber, and the photographic processing.

Most sampled images are approximated by equally spaced samples arranged in the form of an NxN square grid. Such a grid or mask can be easily manipulated in a 3x3 minimum size, as shown in Table 1 [8]. When a pixel belongs to an edge there are four possible directions within a 3x3 neighborhood.

The primary idea underlying any edge detection method is the subtraction of pixels (or groups of pixels) belonging to the same small area. This is usually implemented spatially by convolving the image with a suitable mask that

Photograph 1. Second Edition An Introduction to Programming and Problem-Solving with PASCAL.

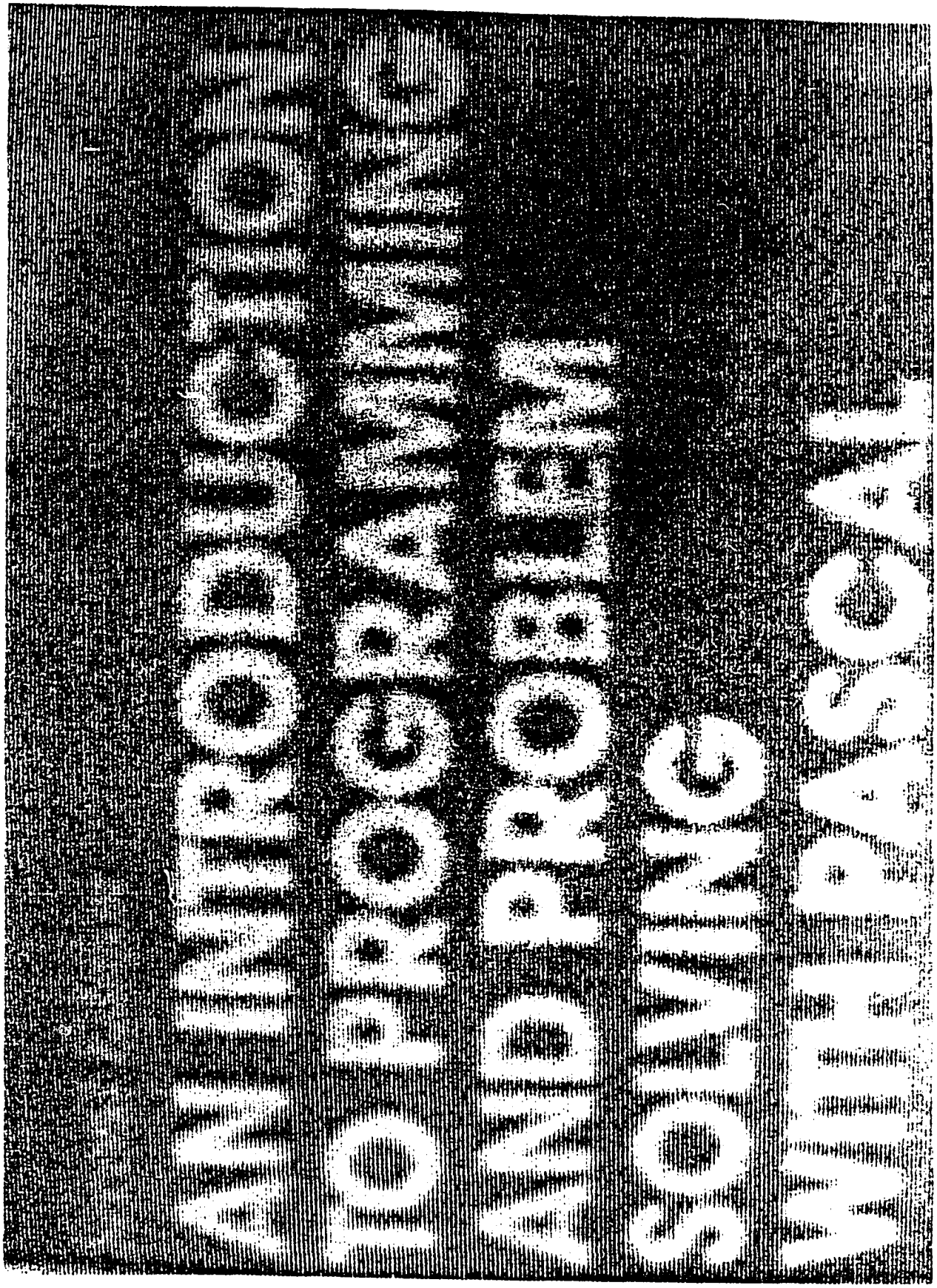


Table 1. Properties of popular edge detectors

Characteristics	Sobel	Prewitt	Kirsh	Laplace
Size of mask	3x3	3x3	3x3	3x3
Relative number of multiplications	2	2	8	1
Threshold level affects edge thickness	yes	yes	yes	no
Edge details	low	low	low	high
Sensitivity to noise	medium	medium	medium	medium
Application on most kind of edges	yes	yes	yes	yes
First derivative available	yes	yes	yes	no
Second derivative available	no	no	no	yes
Edge direction available	yes	yes	yes	no

has positive and negative elements arranged according to the presumed direction of the edge being detected.

The level of the threshold depends on the signal-to-noise (s-n) ratio of the image and the edges of interest. High values of this ratio tend to produce thin noise-free edges; however, a considerable amount of fine detail may be lost. On the contrary, low s-n ratio values retain fine details, but noise effects and background variations are prominent. The principal edges are also thickened [3].

The properties of four popular edge detection methods, namely, Sobel, Prewitt, Kirsch, and Laplace are presented analytically in Table 1 [8]. Among these basic methods, only the first three and some variations of them were implemented for this study. Procedures were written to process the same image that was used for the hardware edge detection method.

In order to form a visual basis of comparison for the effectiveness of the method used, the photographs were taken with a processed window.

A detailed description of the software edge detection methods follows, with attention given in each case to the same two-dimensional image used for demonstrating the properties of the Hardware and Software Edge Detection Methods.

It is useful to observe the time that each edge

detection method takes to process the same image, its sensitivity to noise, and the edge details that each one displays. The procedures are written in Turbo Pascal and are given in the Appendix.

A. Prewitt Edge Detection

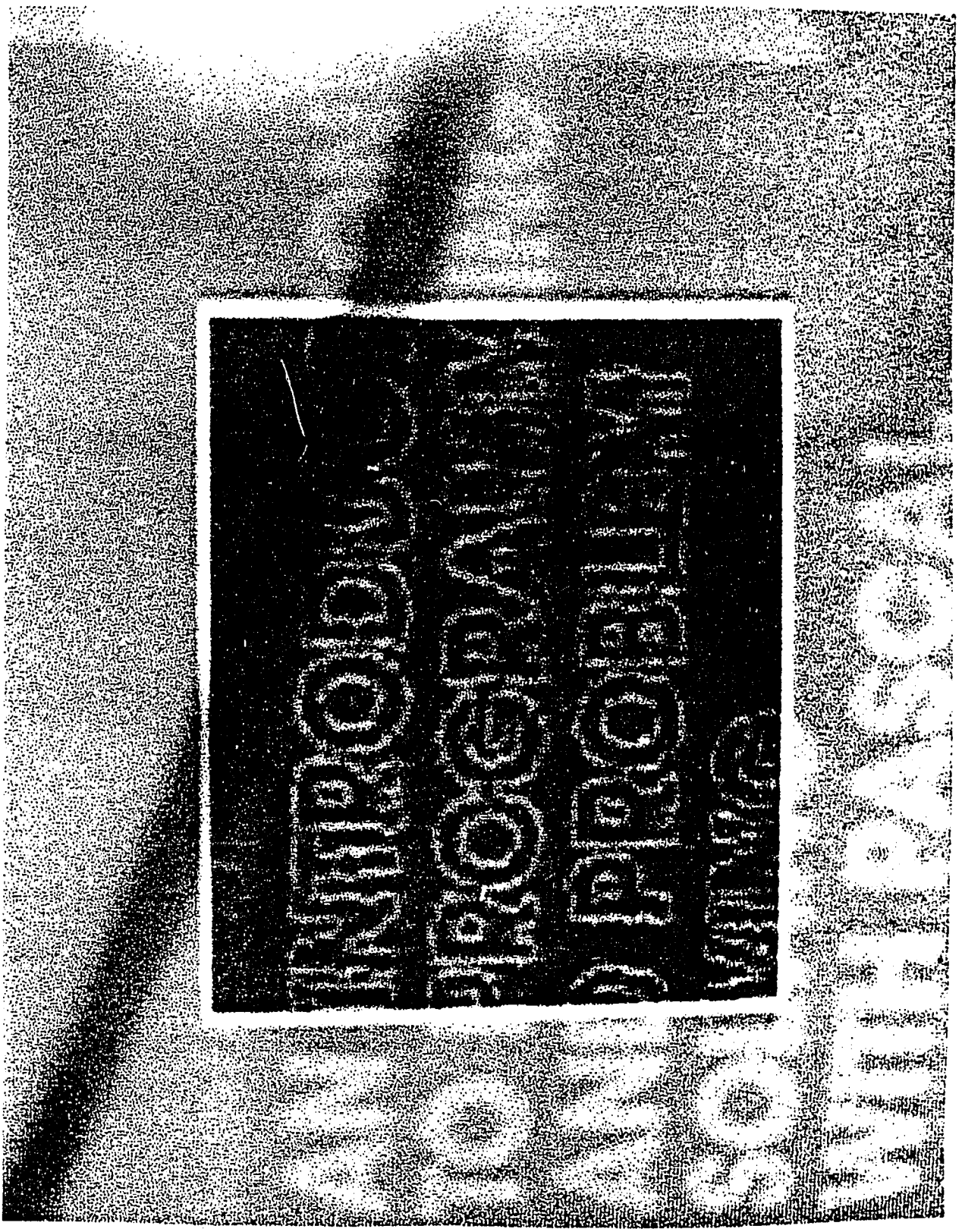
Prewitt suggested [3] two masks that detect horizontal and vertical edges.

$$Py = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad Px = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

These two operators (masks) are applied to each pixel (convolution computation) and the greatest of the two results is chosen as the output. It should be noted that when these operators are applied to the image, the local mean value of the image is computed on both sides of the current pixel along the direction of the detected edge and then subtracted. The stronger the existing edge, the greater the result. Therefore, the detected edge in the output image is more prominent. The slowly varying background is expected to have very low values at the output image, appearing almost black. Due to the existence of noise, the resulting background is not always as smooth as one might expect.

Photograph 2 shows the image resulting from applying

Photograph 2. Prewitt Edge Detection



the procedure Prewitt2 (see Appendix) to a window of the image shown in Photograph 1. The background is satisfactorily black and the edges are unique to the "suppressing nonmaxima" technique. As a consequence of the form of the masks, the detected edges follow very closely the contour of the letters, even though there is a strong preference to horizontal and vertical directions. The pixels that seem to depart from the apparently vertical edges (for example, the letter T of the word INTRODUCTION) follow the real contour of the letter as it was originally recorded by the camera. Close examination of the image in Photograph 1 reveals that noise has altered the straight edges of the original image.

Procedure Prewitt2 took 38 seconds to produce the image shown in Photograph 2, using an HP Vectra Computer (three times faster than the IBM AT Microcomputer). The associated program for this method is written in Turbo Pascal and is given in the Appendix.

B. Thresholded Prewitt Edge Detection

Prewitt [3] suggested two masks that detect horizontal and vertical edges. The masks are the same as in the previous Prewitt edge detection method.

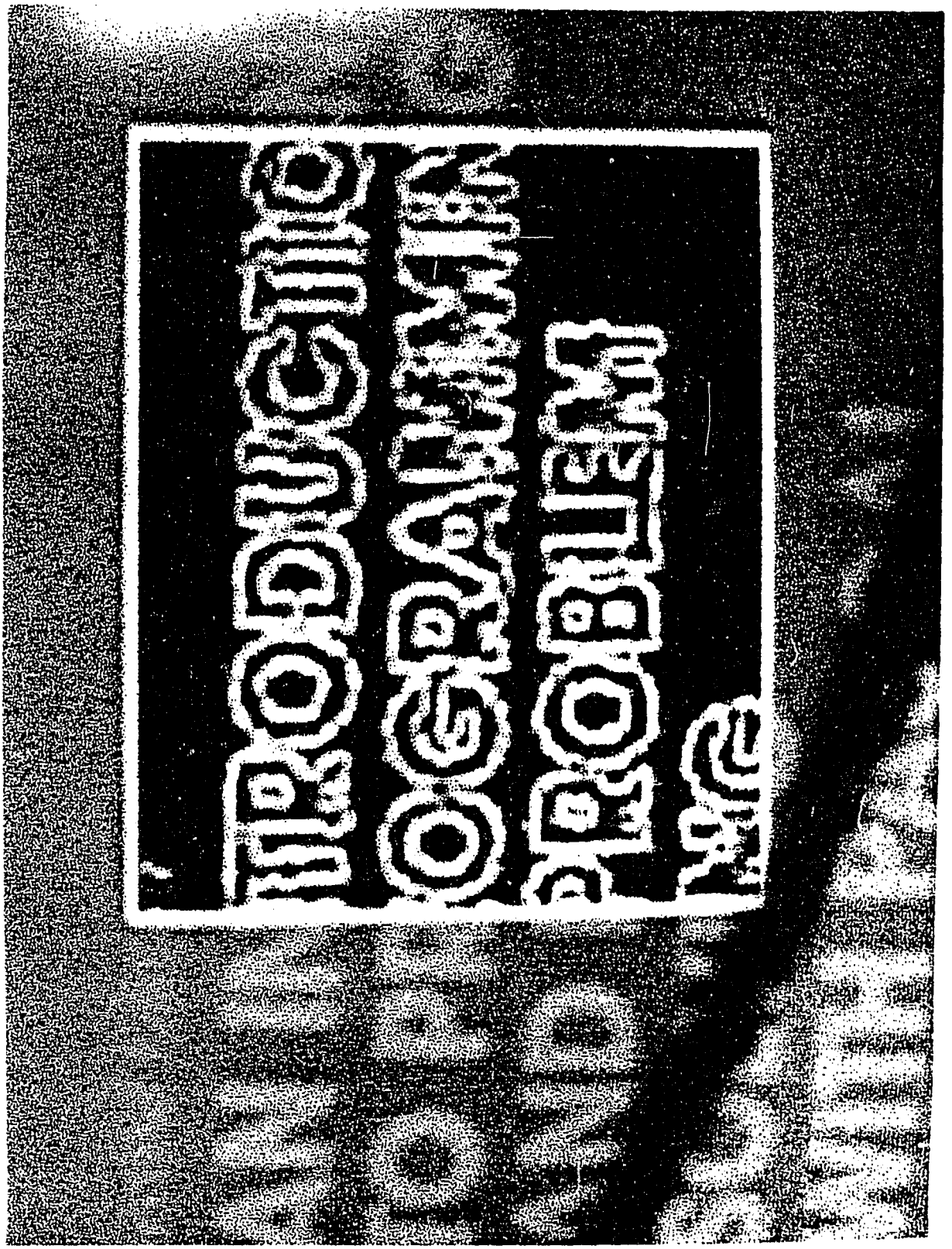
The output is thresholded in order to minimize noise effects. When a threshold is used, there is always a trade-off between fine detail and noise detected as an edge. It

should be noted that when these operators are applied, the local mean value of the image is computed on both sides of the current pixel along the direction of the detected edge and then subtracted. If the difference is significant (i.e., if it exceeds the threshold), an edge is said to exist. Wherever an edge is found to exist, the output can be amplified (multiplied by a constant > 1) so that the edges will be more prominent, while the background is set equal to 0.

Photograph 3 shows the result of Prewitt2a (see Appendix) when applied to a window of the image in Photograph 1. The chosen threshold was equal to 5 and the detected edges were multiplied by 3. Compared to Photograph 2, Photograph 3 has significantly stronger edges and a totally smooth black background. This is, of course, due to thresholding. The edges are actually as thin as those in Photograph 2.

They appear to be thicker because their value is greater and the pixels of the screen have an oblong shape. One should also consider the smearing introduced during the photographic procedure. The edges follow very closely the contour of the letters as in Photograph 2, since it is basically the same method. Horizontal and vertical edges appear to be more prominent as a consequence of the form of the two masks. The pixels that seem to depart from

Photograph 3. Thresholded Prewitt Edge Detection


PROBLEM
PROGRAMMING
PRODUCTION

apparently totally vertical edges follow the exact contour of the letter, as recorded by the camera.

The Prewitt2a procedure implements the method presented below, has a threshold of T=5, and takes 28 seconds to produce the image shown in Photograph 3.

C. Prewitt Edge Matching

Photograph 4 shows the result of applying Prewitt Edge Matching to a window of the image shown in Photograph 1. The edges are considerably thicker than those obtained by the Prewitt or Sobel methods. Photographs 2, 3, and 7 show that the edges are considerably smoother. There is no impression that they consist only of horizontal and vertical segments, but they gently follow the contour of the letters forming perfect curves, often blurring originally sharp corners. This is due, in part, to the fact that their width is more than one pixel.

In the edge matching method, we compute the convolution of the image with 8 different edge detection masks and choose for each pixel as an output the greatest result of the 8 convolutions. The greater the result of the convolution, the stronger or more pronounced the edge in the corresponding direction. The final result can also be thresholded in order to minimize noise effects, i.e., noise detected as an edge.

Prewitt Edge Matching implements this method to produce

the results are shown in Photograph 4 in 16 minutes. The program for this procedure is given in the Appendix.

This method uses 8 masks to detect edges in four possible directions [3] There are 2 masks for each direction (their dimension is usually 3x3). The 8 elements of each mask around the central one constitute two groups: one group of five positive elements and another of 3 negative elements. Each mask can be derived from another by circularly shifting its elements around the central one.

The most commonly used masks are the PREWITT masks:

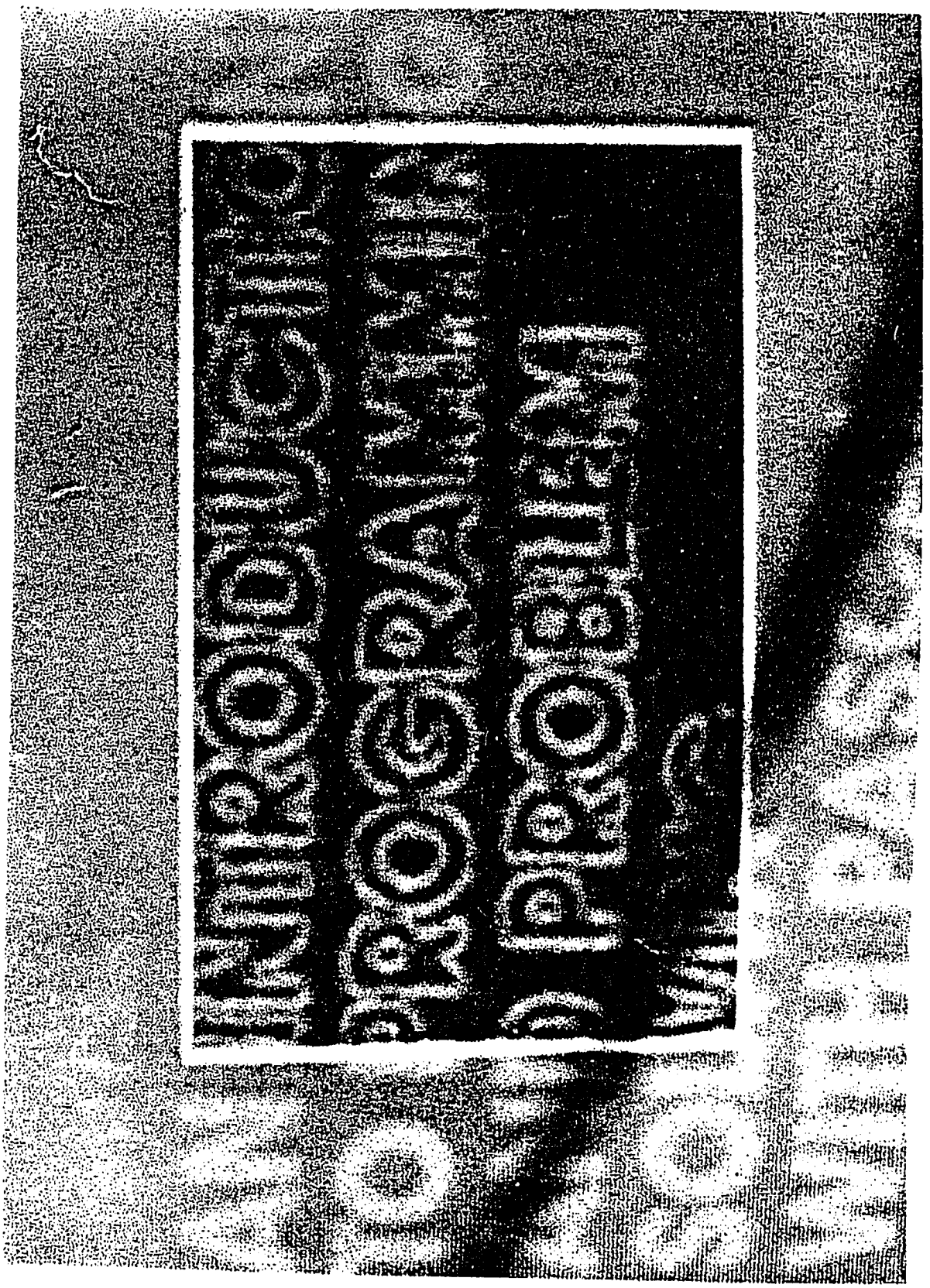
$$\begin{matrix} -1 & -1 & -1 \\ 1 & -2 & 1 \\ 1 & 1 & 1 \end{matrix}$$

The other eight masks can be derived by circularly shifting the elements.

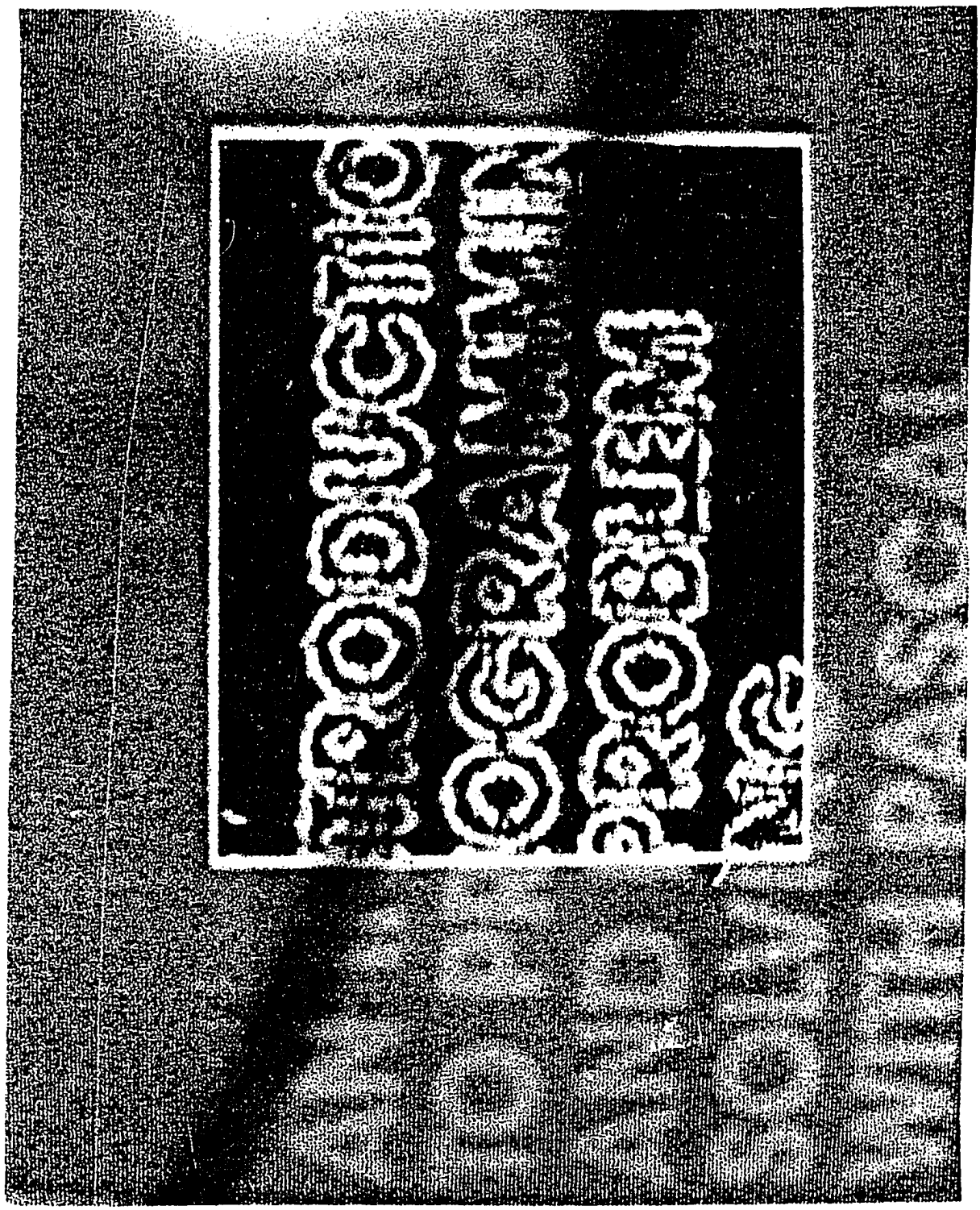
D. Extended Prewitt Edge Detection

Photograph 5 shows the results of applying the Extended Prewitt edge detection procedure to a window of the image shown in Photograph 1. The edges are not as thin as expected. Actually, they are thicker than those obtained by the simple Prewitt or Sobel. Moreover, they are not continuous. They are made up of groups of pixels separated by small spaces. Due to smearing introduced during the photographic procedure, it is not easy to observe this

Photograph 4. Prewitt Edge Matching


INTRODUCTION
PROGRAM
PROGRAM

Photograph 5. Extended Prewitt Edge Detection



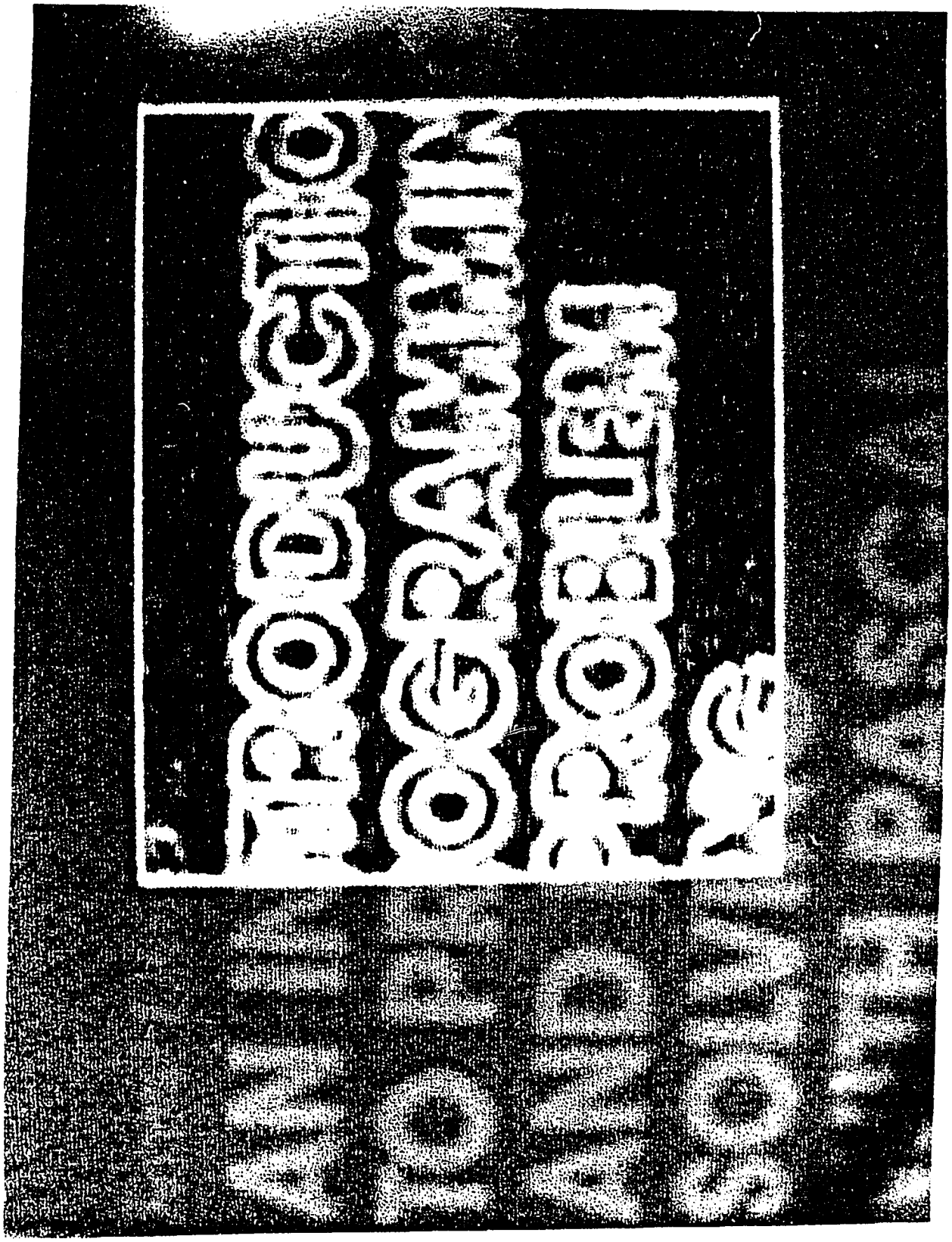
separation in the photograph. Still, one may consider Photograph 5 more pleasing to the eye than Photographs 3 or 7 because the edges follow the curves more smoothly and are not dominated by strictly horizontal and vertical parts as in Photographs 3 or 7. As far as noise is concerned, the Extended Prewitt [12] method seems to have a similar performance than that of the simple Prewitt method and a better performance than the Sobel method. It should be noted that the threshold used in Photograph 5 was equal to 5 as was in Photographs 3 and 7.

The Extended Prewitt procedure processed the image of Photograph 1 in 27 minutes and 30 second. The result is shown in a window of Photograph 5. The program is listed in the Appendix.

E. Kirsch Edge Matching

Photograph 6 shows the results of applying the Kirsch Edge Matching method to a window of the image shown in Photograph 1. The edges are considerably thicker than those obtained by the Prewitt or Sobel methods. Photographs 2, 3, and 7 show that the edges are considerably smoother. There is an impression that they consist of horizontal and vertical parts, but they gently follow the contour of the letters forming perfect curves, often blurring originally sharp corners. This is due, in part, to the fact that their width is more than one pixel. The background is not smooth,

Photograph 6. Kirsch Edge Matching


PRODUCTION
PROGRAMMING
PROBLEM

implying that the method needs thresholding. Tests have proven that indeed the existence of a threshold drastically enhances the performance of the Kirsch Edge Matching method when noise exists.

The Kirsch Edge Matching procedure took 14 minutes and 25 seconds to produce the result that appears in Photograph 6 from the original processed image shown in Photograph 1. The programs for this procedure are given in the Appendix.

Among the commonly used masks are the KIRSCH mask:

$$\begin{matrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{matrix}$$

The Kirsch Edge Matching method computes the convolution of the image with 8 different edge detection masks and chooses for each pixel the greatest result of the 8 convolutions as an output [13]. The level of the threshold depends on the s-n ratio of the image and the edges of interest. High values tend to produce thin noise-free edges, but a considerable amount of fine detail might be lost. On the contrary, low values retain fine detail, but noise effects and background variations are prominent as well. The main edges are also thicker.

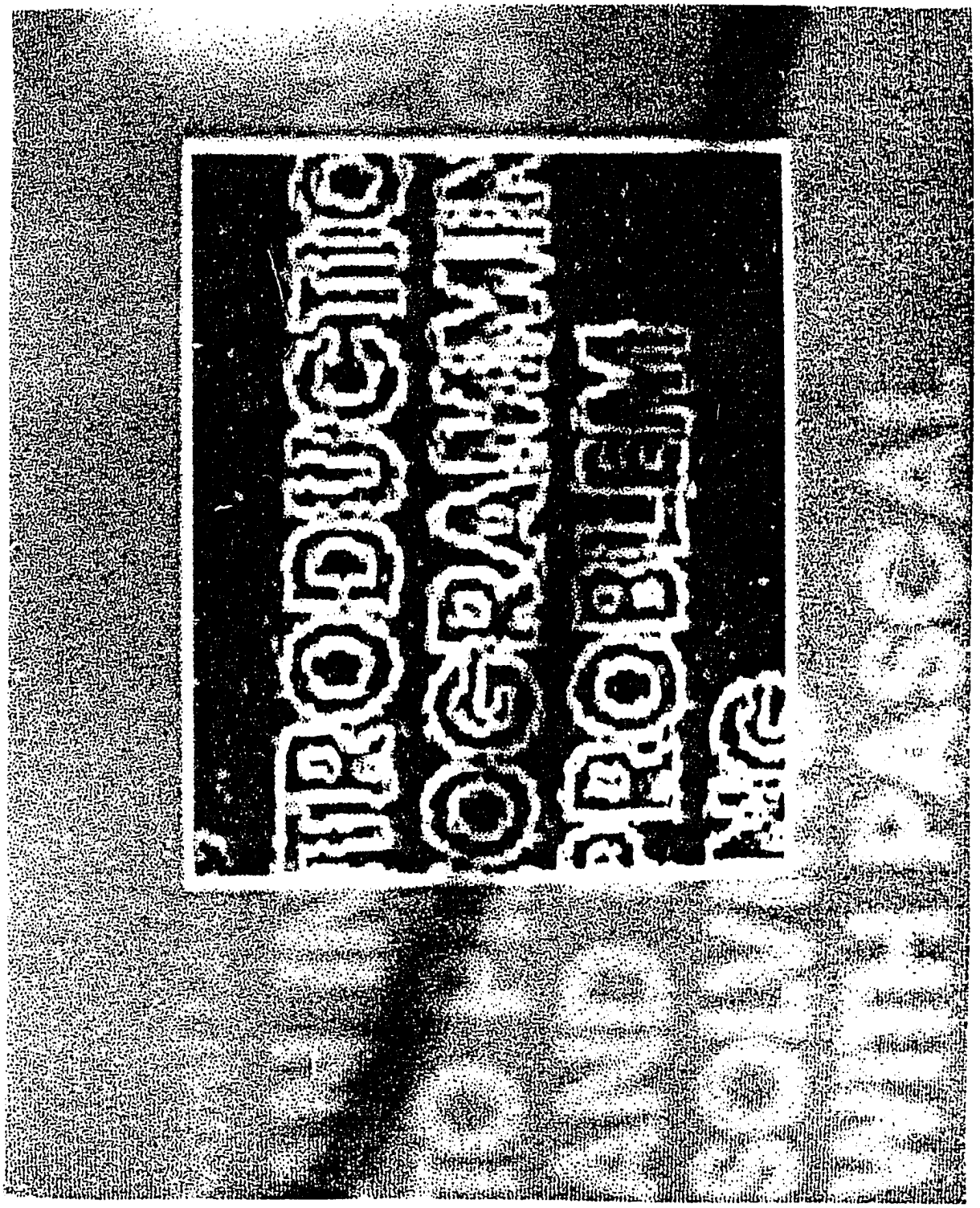
F. Sobel Edge Detection

Photograph 7 shows the results of applying the Sobel Edge Detection method to the image shown in Photograph 1. The result is similar to those obtained by the Prewitt2 and Prewitt2a methods, but are not identical. The edges are as thin as those in Photographs 2 and 3, even though they may appear thicker than those in Photograph 3. This is because their value has been amplified and the pixels of the screen have an oblong shape looking larger than they actually are when they have values.

Horizontal and vertical edges are more clearly detected as a consequence of the shape of the two used masks. Even though the same threshold [5] was used and the detected edges were multiplied by the same constant [3] as in Photograph 3, the Sobel masks seem to be more subject to noise effects than the Prewitt masks. The background is not as smooth as in Photograph 3, where thresholding was also used. The edge pixels appear at places where the contour of the letters has been disfigured form noise during the image grabbing.

Procedures Sobel12a implements the method presented here. Its execution was 42 seconds for the full image and the result (in the window) is shown in Photograph 7. The program for the Sobel Edge Detection method is given in the Appendix.

Photograph 7. Sobel Edge Detection


PRODUCTION

Sobel proposed [8] two masks, similar to the Prewitt ones, that detect horizontal and vertical edges:

$$Sy = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad Sx = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Sobel more heavily weighs the pixels next to the central one at a direction perpendicular to the one detected, as compared to Prewitt. These two operators (masks) are applied to each pixel (convolution computation) and the greater of the two results is chosen as the output. The output can be thresholded in order to minimize noise effects. When a threshold is used, there is always a trade-off between fine detail and noise detected as an edge.

These operators are applied by computing the local mean value of the image on both sides of the current pixel along the direction of the detected edge and then subtracting them. If the difference is significant (i.e., if it exceeds the threshold), an edge is said to exist.

IV. DESIGN AND IMPLEMENTATION OF FRAME GRABBER

A. General Characteristics

The Grabber/Display card is one of the three cards in the Image Processing system. It communicates with the other two cards, the Central Processing Unit card (CPU) and the Peripheral Control card, via a system bus called the T-bus. The card accepts an external composite video signal, digitizes the signal, and then stores it in binary form. It can display the image on an RGB or a black and white analog monitor. It can also transfer or accept an image to or from the CPU card for processing. The transfer occurs one frame at a time, although it is possible to transfer smaller fragments. The image has a resolution of 256 by 256 pixels, each pixel having 64 gray levels. Pseudo coloring is also supported in the card, displaying eight colors simultaneously. The card has the capacity of keeping four images on board, giving the capability of switching very fast from one image to the other and producing such effects as instant comparison and/or animation.

B. Block Description

The block diagram of the Grabber/Display card is shown in Figure 2. It consists of four basic sections: Memory, Display, Grabber, and Control.

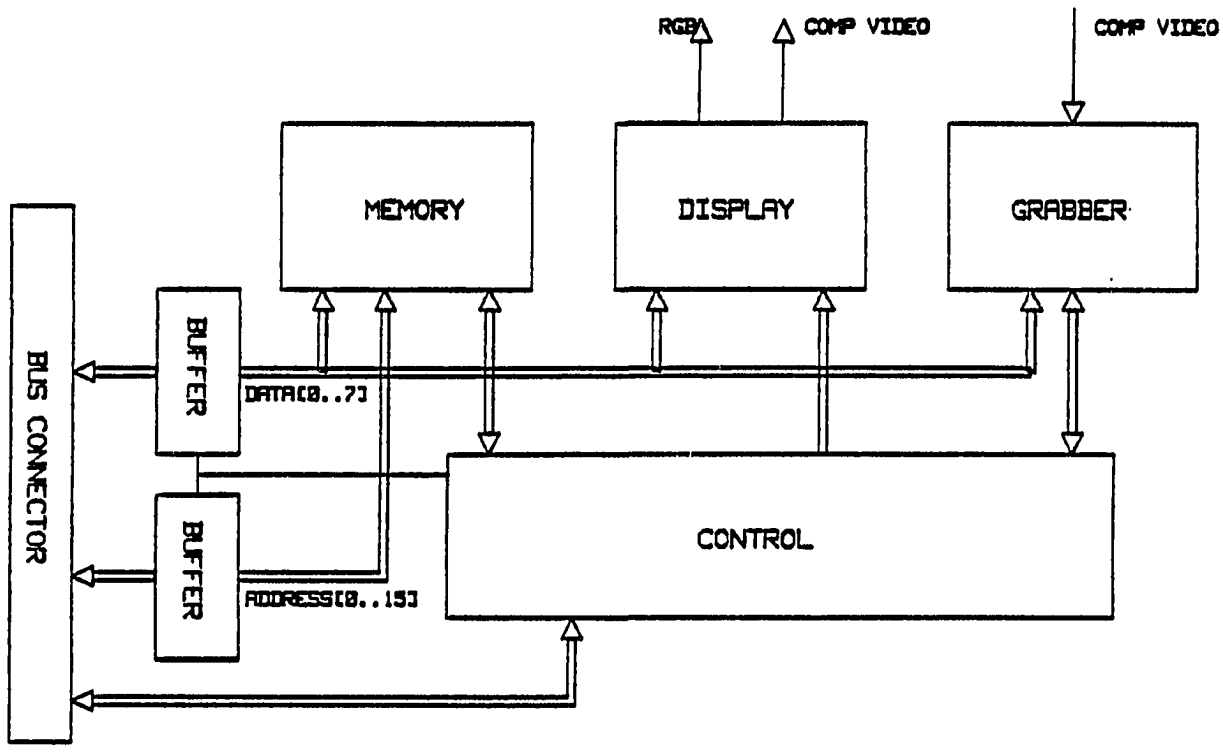


Figure 2. Grabber/display block diagram

1. Memory section

The memory section is the physical location where the four images are stored. This memory can be accessed from the Display section, the Grabber section, and the CPU card.

2. Display section

The Display section is responsible for taking the selected image from the Memory section and displaying it on the color and/or black and white monitor. The CPU card can also access an LUT (Look-up Table). The LUT is responsible for transforming each gray scale to another gray scale value. This feature can be used to instantly generate a reverse video display, an enhanced contrast video, or even

correct non-uniform lighting conditions due to lens defects.

3. Grabber

The Grabber section is responsible for digitizing the composite video input signal frame-by-frame. It then stores it in the selected memory bank.

4. Control section

The Control section finally coordinates the control signals to and from the CPU card, the Memory, the Display, and the Grabber sections.

C. Grabber Modules Description

The detailed description for each one of the modules described above is given in the following paragraphs.

1. Memory section

Figure 3 shows the detailed block diagram of the memory. The four logical banks are implemented through eight memory chips, 32K bytes each. Each bank uses two of the memory chips, giving a total of 64K bytes for each image (256 by 256 by 8 gray levels). The bank selected is determined by the control register in the Control section. The control logic asserts one of the CS signals according to the chip selected.

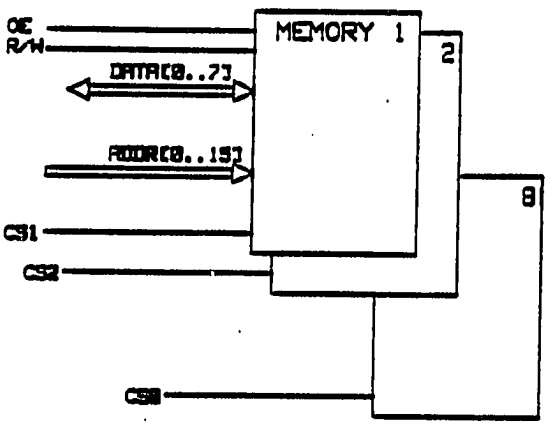


Figure 3. Memory block diagram

2. Display section

Figure 4 shows the detailed block diagram of the Display section. Each module is described next.

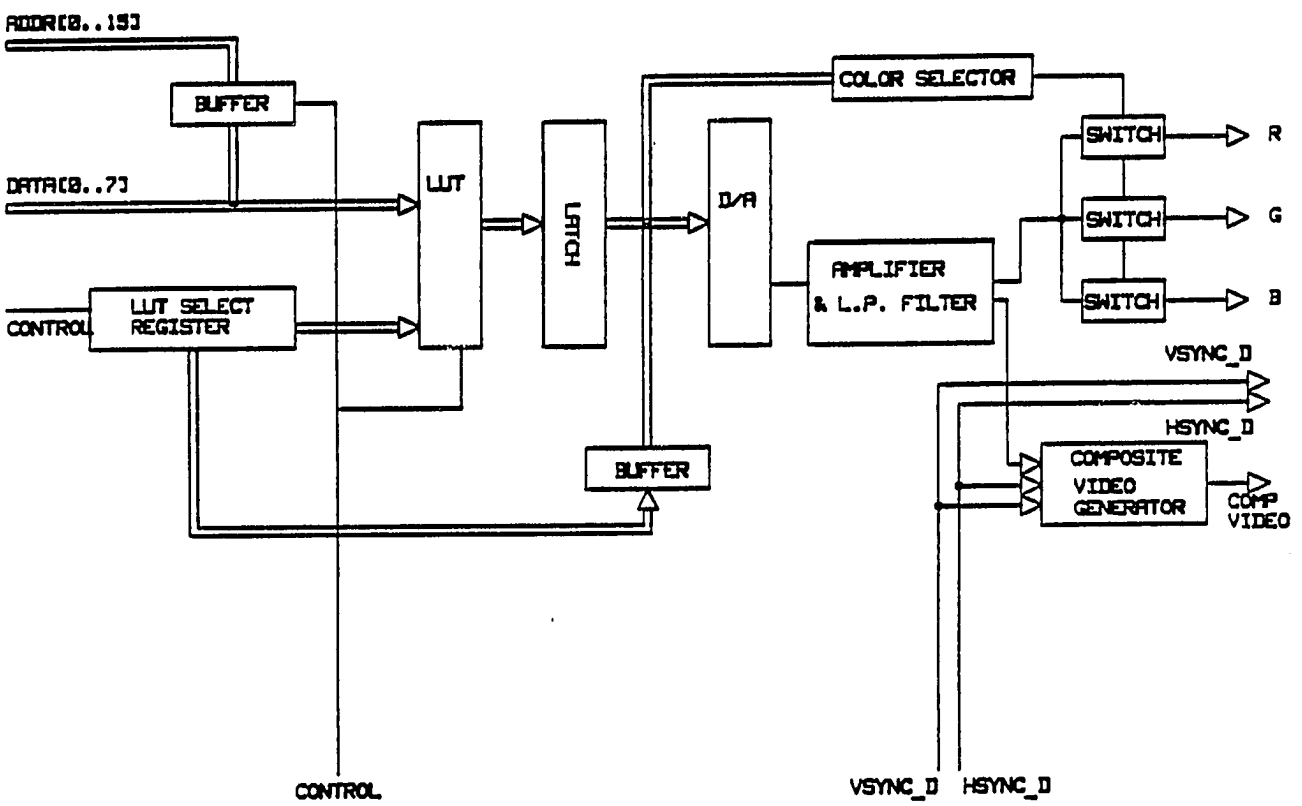


Figure 4. Display block diagram

a. Look-Up Table (LUT) The Look-up table is implemented in the form of a memory chip 2K bytes in size. The 6 lower address bits (A0..A5) are driven from the DATA bus, translating the 64 gray levels to another set of 64 gray scales. The 5 higher address bits (A6..A10) are driven from the LUT SELECT REGISTER, selecting one out of 32 possible look-up-tables. All of the 32 LUTs are physically located in the same 2K-byte memory chip. The memory can be either RAM or ROM. If it is RAM, the CPU must initialize the Look-up tables needed for the application.

b. LUT Select Register The LUT select register is a write-only register. It can be accessed from the CPU card in order to select a particular Look-up table. The I/O space address of the register is at $320.

c. Latch The latch provides timing isolation between the Memory-LUT path and the D/A module. A new value is latched to every new memory cycle (5 MHz), providing a stable input to the Digital-to-Analog module.

The output of the latch is tri-stated during the blanking signal and pull-up resistors create a value sent to the D/A converter which results in a black display.

d. Digital-to-analog converter The digital-to-analog converter translates the digital input to an analog output voltage. The analog voltage is low-pass filtered at

about 5 MHz.

e. Amplifier and color selector The amplifier drives the red, green, and blue guns of the analog RGB monitor through analog switches. It also drives the composite video generator circuit.

The values of the three least significant bits from the LATCH (D[0..1]) control the analog switches for producing the displayed color. Since 3 bits are used, we have 8 possible color combinations.

f. Composite video generator The composite video generator takes as input the analog video signal and the HSYNC_D and VSYNC_D signals and produces a composite signal for use with composite black and white monitors. The composite video signal conforms to the European Standard.

3. Grabber section

Figure 5 shows the detailed block diagram of the Grabber section. Each consists of the module as described below.

a. Limiter The limiter protects the rest of the circuitry. It limits the maximum input to a predefined level range.

b. Low pass filter The low pass filter blocks out any frequencies above 2.5 MHz to avoid aliasing of sampled

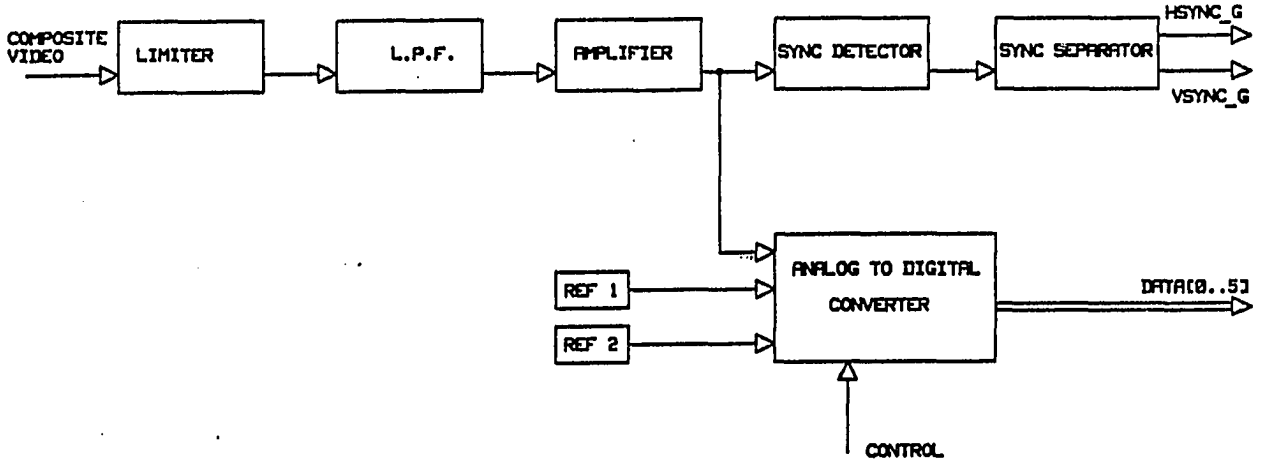


Figure 5. Grabber block diagram

signal. It also extracts the color information from the composite signal.

c. Amplifier and sync separator The video amplifier stage has a gain of 1. The sync separator separates the synchronization signals from the video information. It produces a composite sync signal containing both the horizontal and vertical sync signals.

d. Sync separator The sync separator splits the composite sync signal into two syn signals, the horizontal sync (HSYNC_D) and the vertical sync (VSYNC_D).

e. Analog-to-digital converter The analog-to-digital converter is a very fast flash A/D converter producing a stable digital output on every memory cycle (5 MHz).

4. Control section

Figure 6 shows the detailed block diagram of the Control section. Each module is described below.

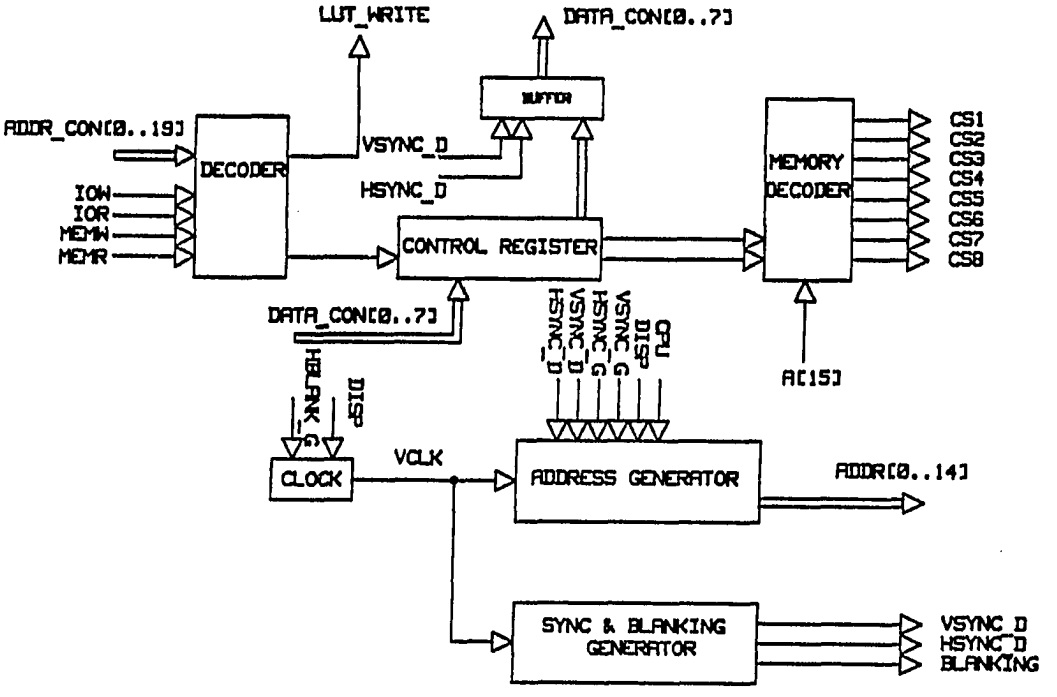


Figure 6. Control block diagram

a. Decoder The decoder decodes the addresses from the system bus to identify when the card is addressed by the CPU. The addresses selected are for the CONTROL REGISTER and for the LUT SELECT REGISTER.

b. Memory decoder The memory decoder selects one of the eight image memory chips on the board. The bank selected is determined by bits 4 and 5 in the CONTROL REGISTER.

c. Clock The clock circuit provides a 20 MHz and a 5 MHz signal to the rest of the control circuitry. The clock is also synchronized via the external signals HBLANK_G and DISP in order to be in step with the incoming video signal during the grabbing phase.

d. Address generator This module provides the addresses to the memory section during the grabbing and the display phases of the board. The address generator is in sync with either the incoming video signal's syncs (HSYNC_G and VSYNC_G) or with the outgoing video signal's syncs (HSYNC_D and VSYNC_D).

e. Sync and blanking generator This unit generates the output video's sync signals VSYNC_D and HSYNC_D. It also generates a blanking signal used to blank off the display during horizontal and vertical retraces.

D. Frame Grabber Detailed Description

This section will give a more detailed description of the components. The chips are also given U numbers for reference. The placement diagram is shown in Figure 7.

1. Bus isolation

U1, U2, and U3 provide isolation between the system bus and the grabber card. U1 (LS245) is a bidirectional data buffer. U2 and U3 (LS244) are unidirectional address

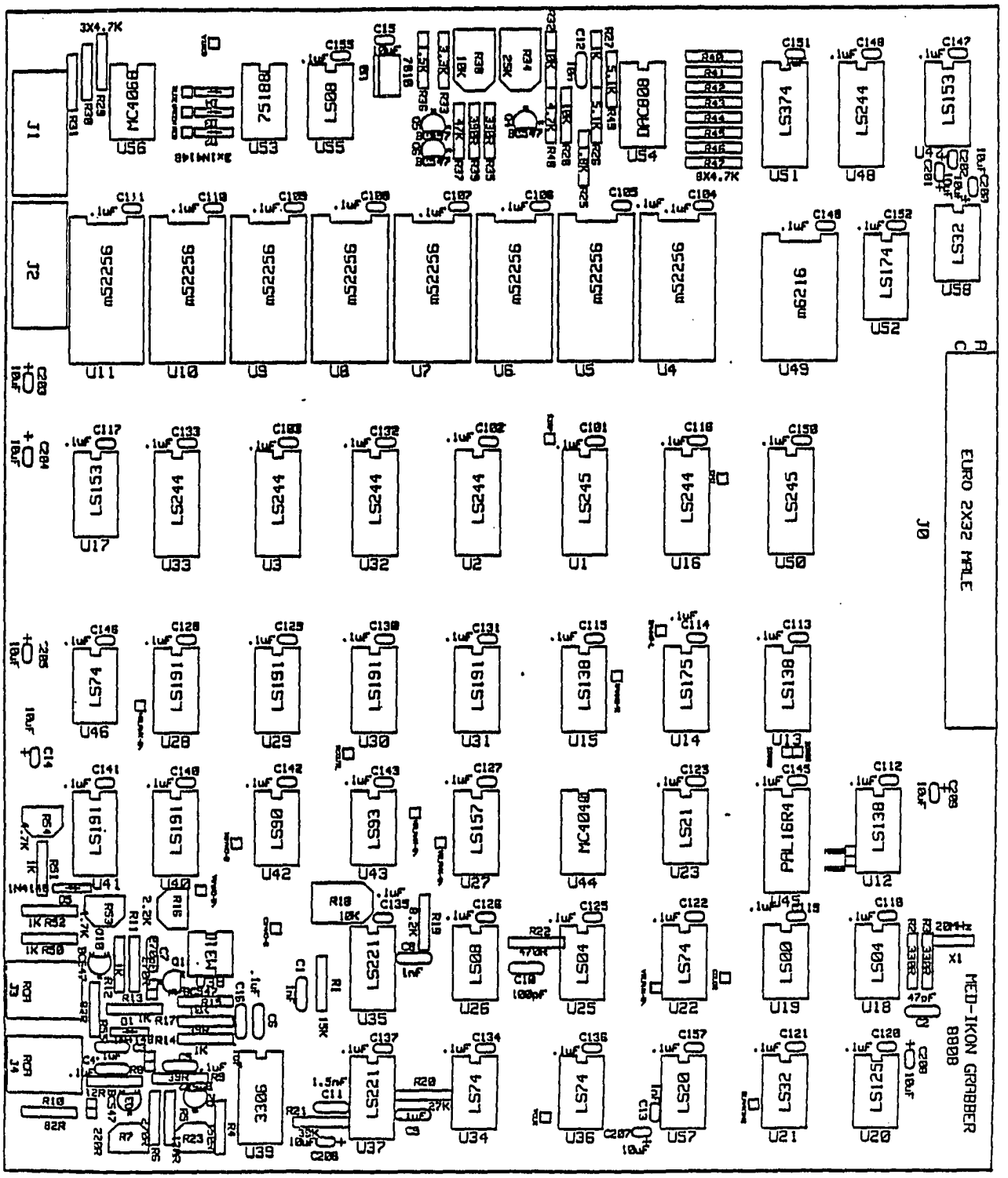


Figure 7. Component placement diagram

buffers. The output buffered buses are the DATA and ADDRESS buses, respectively. The buffers are enabled when access to the card is indicated by the memory decoding chips and the CONTROL REGISTER.

2. Memory section

U4 through U11 are static 32K byte memory elements that can hold a total of four images. U4 holds the upper half and U5 the lower half of the first bank. Similarly, the rest of the ICs hold three more images. The ADDRESS and DATA buses are connected in parallel to all chips. The selection between the lower and upper image area is done with A15. The appropriate bank is selected according to the value of Bits 4 and 5 in the CONTROL REGISTER. The memory bank and chip selection is done with U15 (LS138). The image is stored in memory sequentially with the lower address corresponding to the upper left corner, and the higher address with the bottom lower corner. Adjacent memory locations correspond to adjacent columns in the image.

3. Display section

The data value (intensity value) coming out of the memory enters the Display section in order to be shown on the monitor. The first thing it encounters is the LUT (Look-Up Table), implemented as a 6116 memory element designated as U49. This is a static 2K byte memory. The

DATA bus bits D0..D5 are connected to A0..A5 of the LUT. The rest of the address bits of the LUT (A6..A10) are driven by the LUT SELECTION REGISTER U52 (LS174) in order to select a particular LUT. The intensity value contained in the DATA bus selects one memory address in the LUT. The preprogrammed LUT then outputs another 8 bit intensity value to the data lines of U49. This value is then latched by U51 (LS374) with the falling edge of VCLK (the pixel clock) in order to provide a stable signal to the D/A. The D/A U54 (DAC806) transforms this 8 bit value into an analog voltage. Some passive components on the D/A (C12 and R27) are responsible for low-pass filtering the output at about 5 MHz. The analog voltage (VIDEO) is then fed to the R, G, and B outputs via three analog switches (U56 4066) and to a composite video generator.

The analog switches are controlled from the three least significant lines of the latch (U51) if the COLOR signal is asserted. U55 (75188) is used as a voltage shifter in order to correctly control the analog switches. Diodes D2, D3, and D4 do not allow the negative voltages (normally output by the voltage shifter U53) to be present, since the analog switches do not use bipolar voltages. The R, G, and B outputs can be used then to drive any analog RGB monitor.

An amplifier used to drive the analog BW monitor has an adjustable gain of 2 to 5 (trimmer R34). The offset

measured at the base of Q4 must be adjusted to about 5 volts. The ac component of the video (emitter of Q6) must be about 4 volts peak-to-peak. The tri-state buffers U48 (LS244) and U50 (LS245) are used to make the CPU card "see" the LUT for programming purposes.

The composite video generator has two adjustable controls, one for the gain and one for the dc offset. Trimmer R54 controls the gain and must be adjusted to give an ac component of about 1 volt peak-to-peak (RCA J3 connector). Trimmer R53 controls the offset and must be adjusted to give an offset of about 1 volt.

4. Grabber section

The incoming composite video signal (either color or BW) is first limited to a predefined value with Dq and R11, R12. A terminating resistor is added to the circuit, if the video signal is not terminated at any other point (i.e., by another monitor connected in parallel).

If the source video contains color information, then the capacitor C7 must be connected via the jumper in order to form a low-pass filter. Q1 acts as an isolation amplifier. The video signal at this point goes to the A/D and to the sync separation circuitry. The A/D circuit has two adjustments which define the upper and lower digitization level. R23 is adjusted so that the voltage at the emitter of Q2 is equal to the maximum dc value of the

video signal at pin 11 of U39. R7 is adjusted so that the voltage at the emitter of Q3 is equal to the minimum dc value of the video signal at pin 11 of U39 (without taking into account the negative-going sync pulses). The digitized 6 bit value is then available in the DATA bus and is latched by the memory. The other module to which the video information goes is the sync separator circuitry.

First, the composite sync pulses (horizontal and vertical) are separated from the rest of the video information. In order to achieve this separation, the video enters a comparator (U38 LM311) which compares the video signal with a dc value set to about 1 volt (adjusted by R16 and measured at pin 3 of U38). Since only the sync pulses go below this threshold, the output of the comparator (U38 pin 7) contains the composite sync pulses. U35 (LS221) is used to make the horizontal sync pulses longer.

R18 is adjusted so that the negative pulse at pin 4 of U35a is about 8 microseconds long. U37 is used in order to recognize the longer pulses (more than about 10 micro-seconds), which constitute the vertical sync on pin 4 of U37.

5. Control section

The Control section can be separated into three local sub units. The first unit decoded the address bus in order to designate to the other sections which part is addressed.

The second unit contains the control registers that hold the "state" of the machine. The third section generates the addresses used to scan the memory during the grabbing and the displaying phases.

V. HIGH-SPEED EDGE DETECTION METHOD

This section will illustrate a novel method of identifying edges in a raster image using hardware techniques. The principle of operation is based on control of the timing characteristics of the display module of the image processing system.

A detailed description of the display module is shown in Figure 8. This is essential in developing the platform of understanding for the High-Speed Edge Detection method. In this block diagram as we can distinguish nine basic modules.

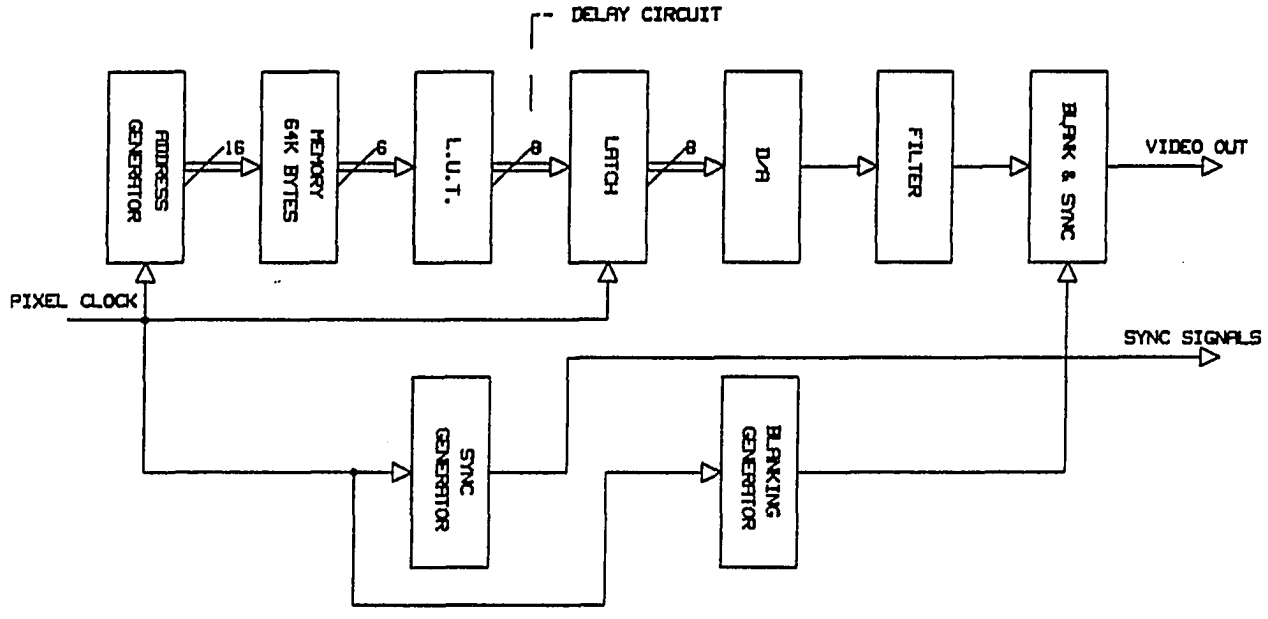


Figure 8. Block diagram of display circuitry

The address generator unit supplies the address of the next pixel to be displayed from the memory. The address vector is sixteen bits wide and has a direct access to 64K bytes of memory. Each memory cell corresponds to a picture element (pixel). The value determines the intensity of the gray level for this picture element.

The memory is organized in an array of 256 by 256 picture elements. Each pixel reserves eight bits of array memory that allows a total of 256 different intensity values, but the A/D converter resolution is only six bits, which limits the effective display gray levels to 64. These 64 gray levels are translated via a Look-Up table (LUT) to 256 other gray level values. The LUT is a translation table which gives an intensity value range from 0 to 255 for every table entry [11]. The intensity value coming out of the LUT is then latched in a synchronous mode with the pixel clock at an effective frequency of 5 MHz.

The latched value then enters the D/A converter which translates the 8-bit vector to an analog voltage. The analog signal is then passed through a low-pass filter in order to remove undesirable high frequencies introduced during the conversion stage. Finally, the signal is combined with the horizontal and vertical sync signals before becoming output to the monitor.

A. Hardware Implementation

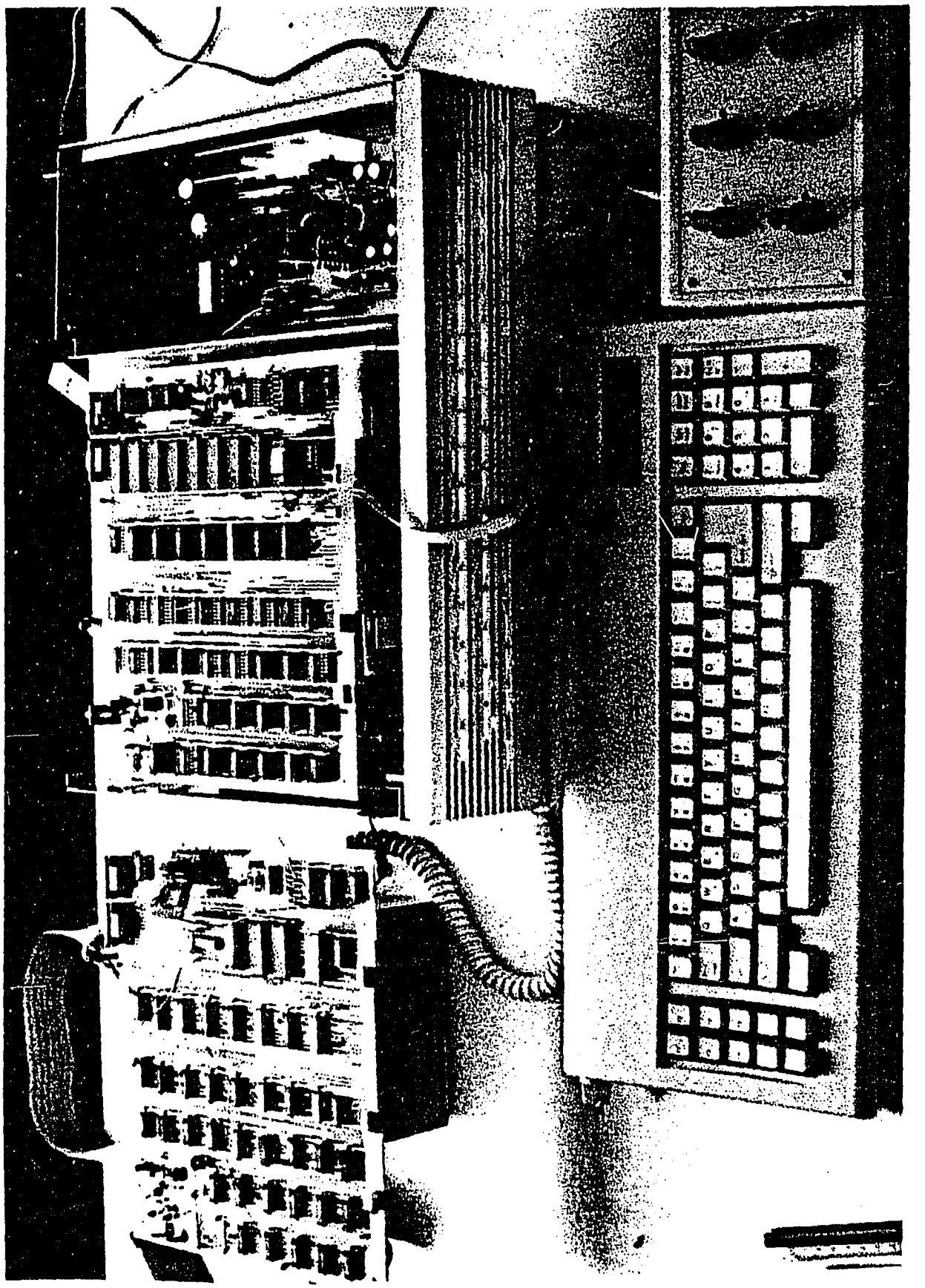
A delay mechanism introduces a timing change to the display module. The delay is applied after the LUT unit and before the latch. The duration of this delay is of the order of 200 nanoseconds and is introduced to one or more of the data lines. The result is a temporary change to the intensity values supplied to the latch, producing either a black or white spot on the monitor. A detailed description of the hardware implementation that changes the timing follows.

An apparatus was developed (Photograph 8) which permits exhaustive control of the amount of the delay that can cause effects in the image. This apparatus consists of six five-position rotary switches. Each switch is connected to each output data line of the LUT.

The five positions of each switch correspond to an equal number of capacitors. By introducing capacitance between a data line and the ground, the instant at which the signal changes state can be controlled. Five cases of different timing characteristics which produce different results by applying the delay to only one data line signal will be described.

The five cases differ from each other by introducing capacitance in constant increments, (0.001 microfarads), with the first case having no delay.

Photograph 8. View of Image Processing System with high-speed edge detection



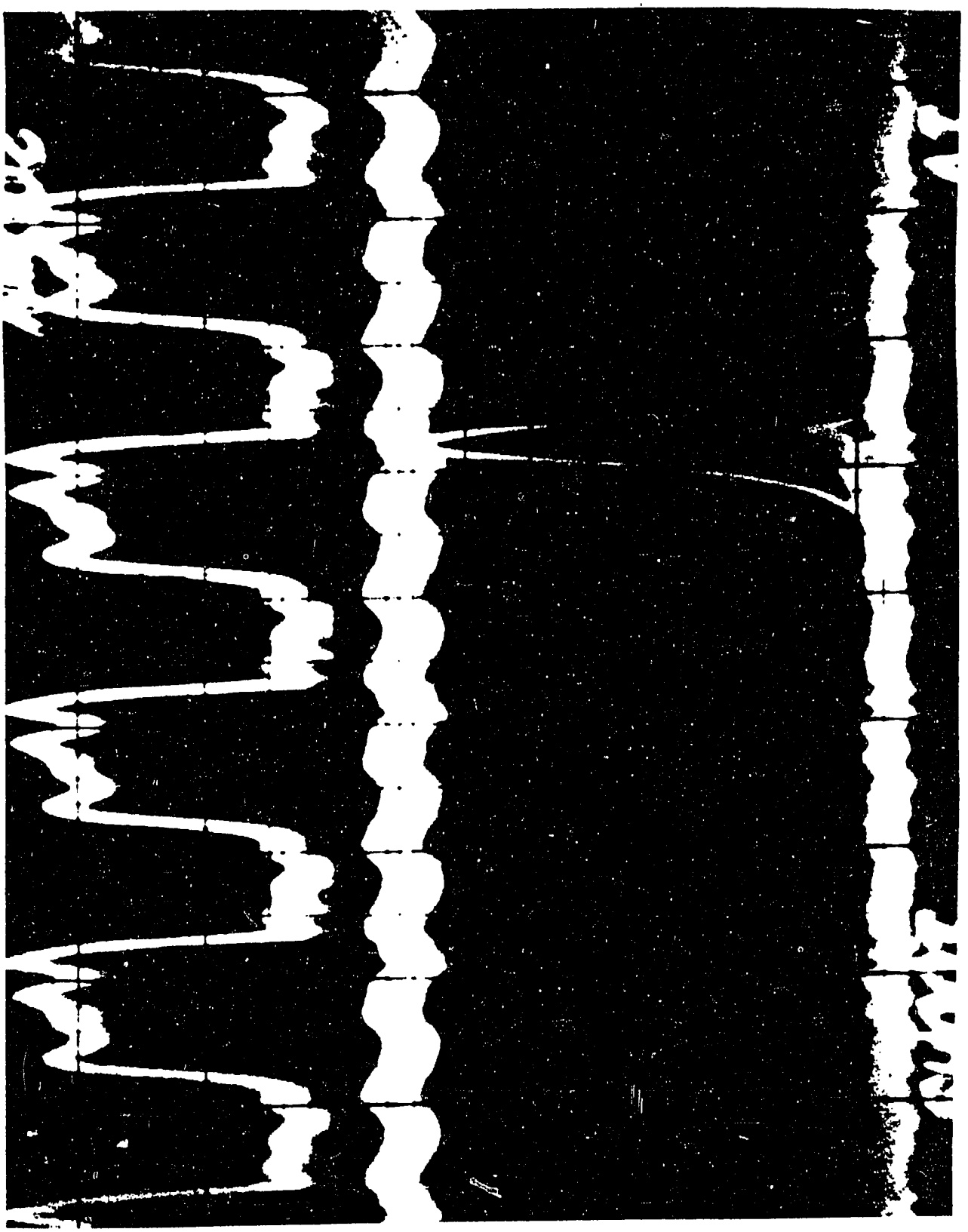
B. Oscilloscope Timing Photographs

Each photograph identifies two traces. The top trace is the data line signal under study. The bottom trace is the pixel clock (5 MHz). Each transition of the data line occurs on the falling edge of the pixel clock. In the first case, shown in Photograph 9, when no delay was introduced, the data line was completely stable, one clock period after the falling edge.

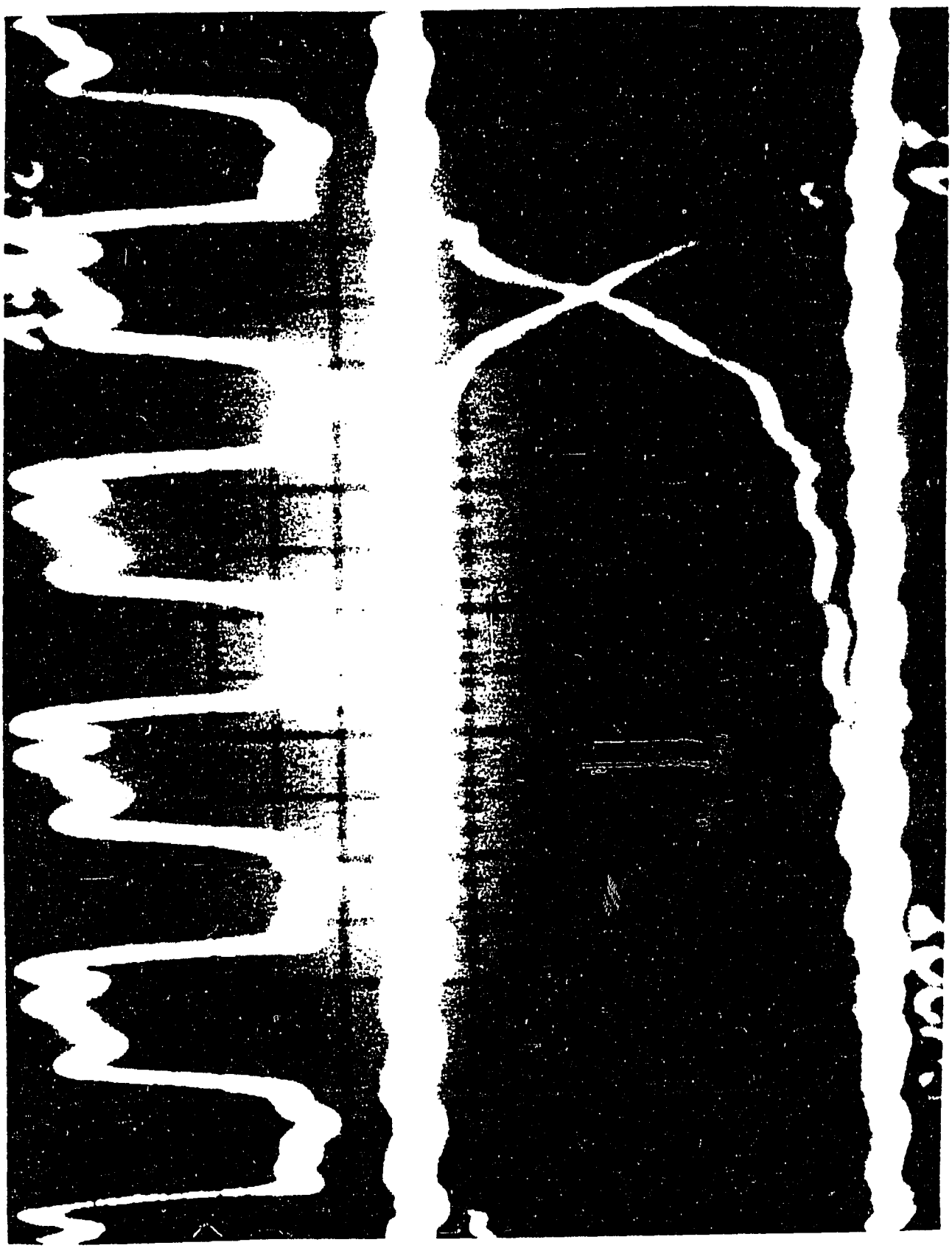
In the second case, shown in Photograph 10, we introduced a 0.001 microfarads capacitor on the data line. The data line reached about 85 percent of its final value when the next falling edge came along. In the third case, shown in Photograph 11, when the capacitor value was doubled (0.002 microfarads), the data line reached about 75 percent of its final value. In the fourth case, shown in Photograph 12, the capacitor value was tripled (0.003 microfarads) and the signal reached 50 percent of its value. In the fifth case, shown in Photograph 13, the capacitor value was quadrupled (0.004 microfarads). As a result, the signal reached about 40 percent of the final value, in one clock period.

Delaying one of the data lines (i.e., D5) as shown in Figure 9 will cause a black spot (pixel) to appear on the monitor. A transition in gray scale intensity from the value of 31 to 32 will cause bits D0 through D4 to change

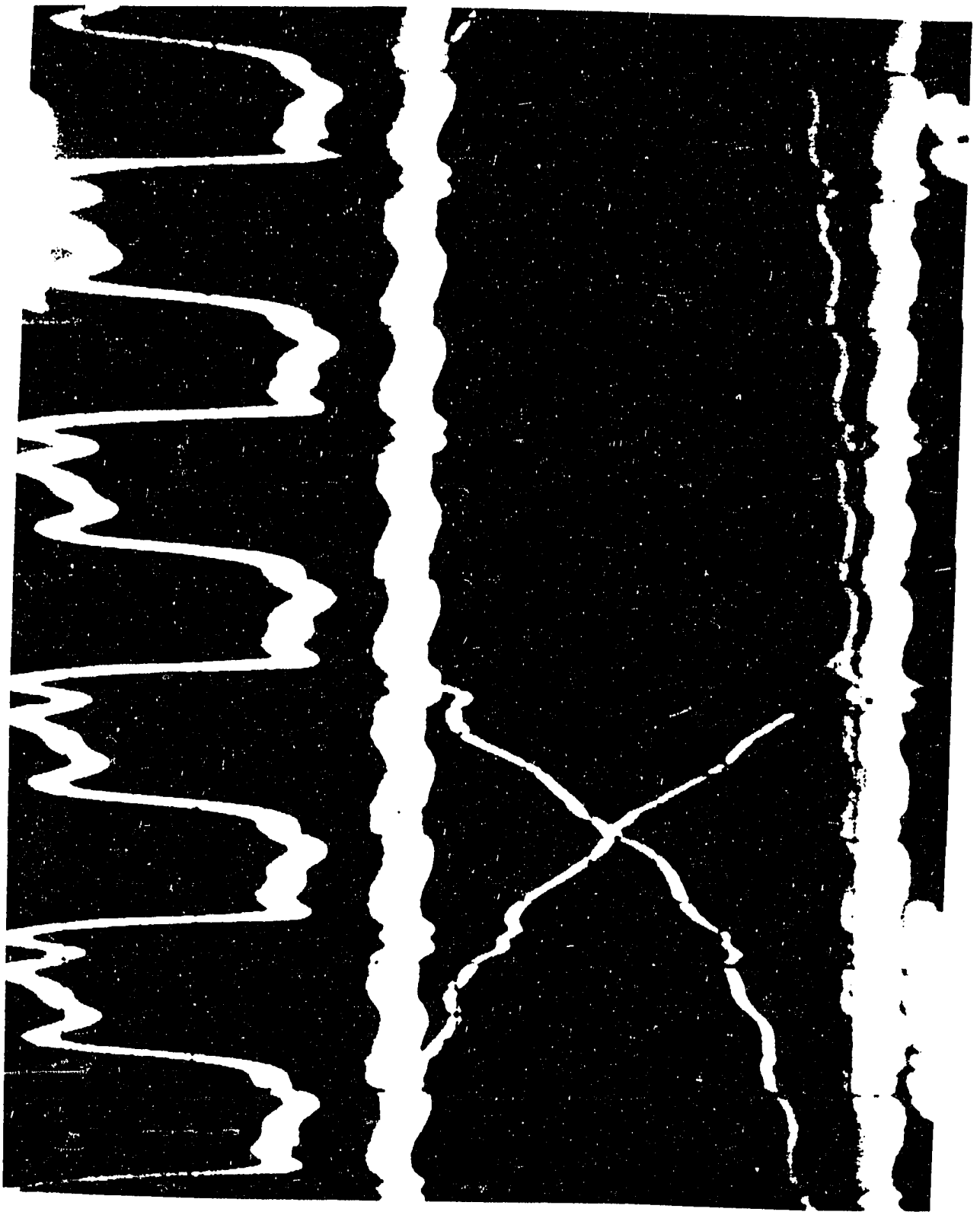
Photograph 9. Oscilloscope picture with no delay



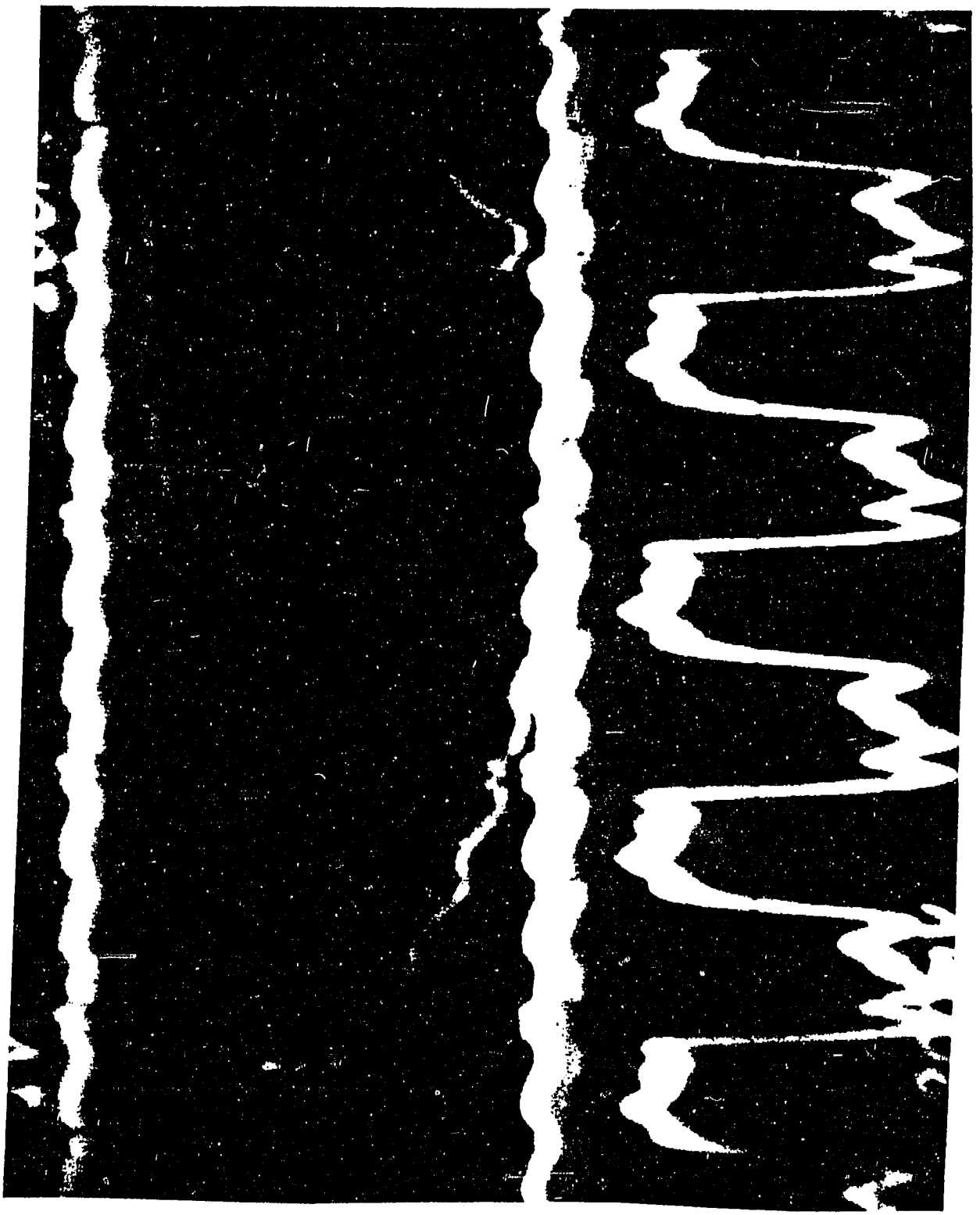
Photograph 10. Oscilloscope picture with capacitance 0.001 μF



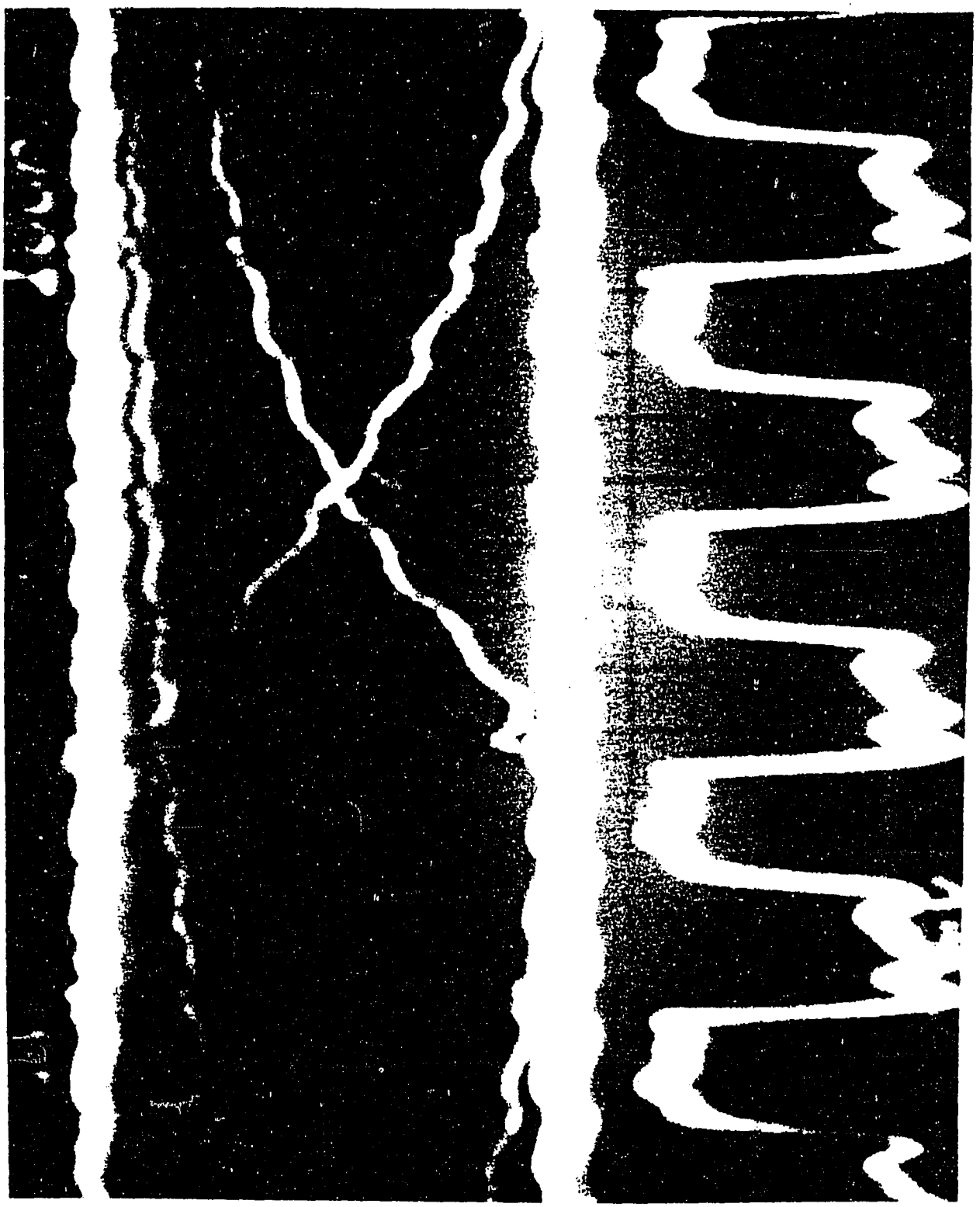
Photograph 11. Oscilloscope picture with capacitance 0.002 μF



Photograph 12. Oscilloscope picture with capacitance 0.003 μF



Photograph 13. Oscilloscope picture with capacitance 0.004 μF



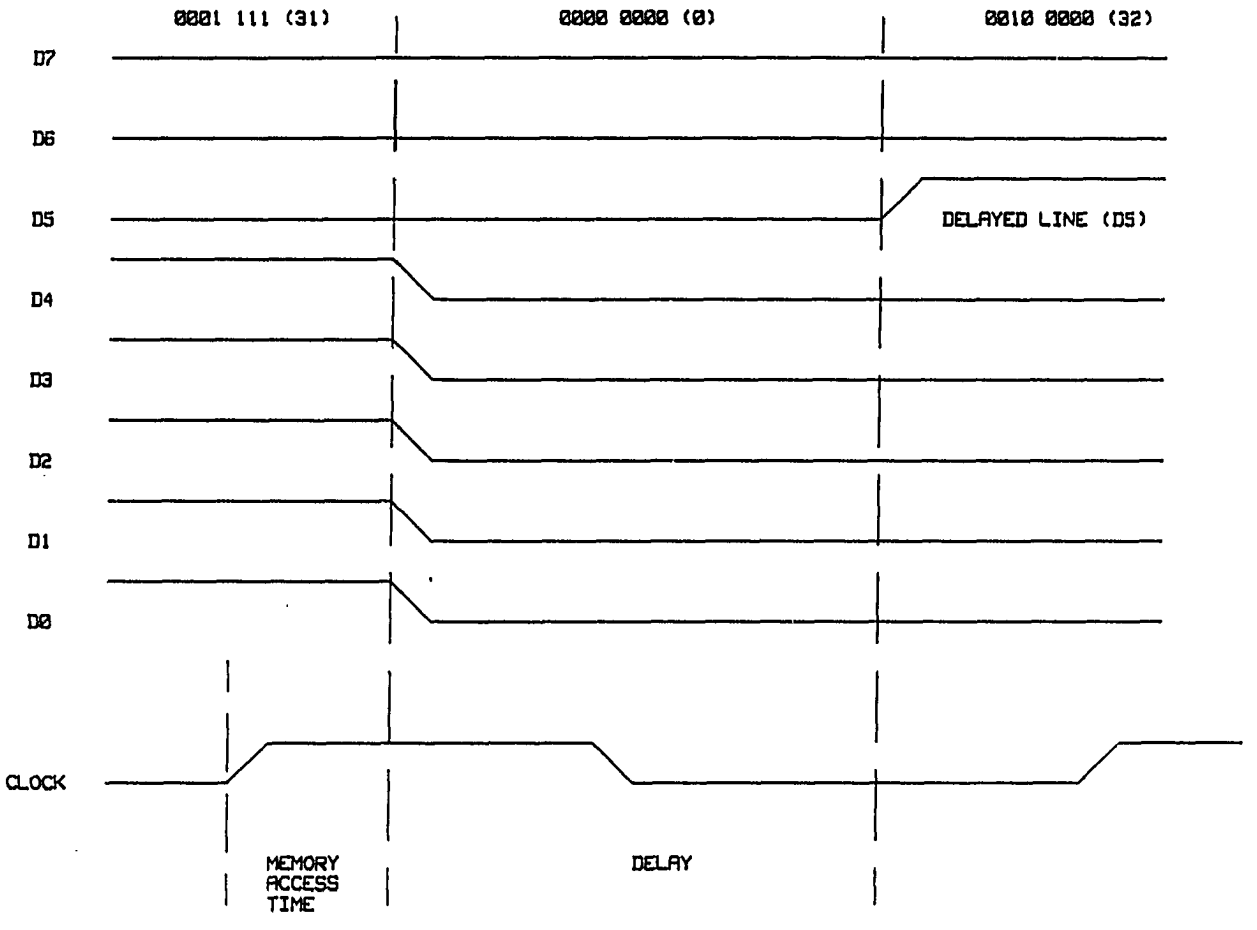


Figure 9. Timing diagram for black spot

from all 1's to all 0's, and bit D5 will change from 0 to 1.

A similar effect will occur during the opposite transition, i.e., from gray scale value 32 to 31. Bits D0 through D4 are initially all 0's and then become all 1's, and D5 should also go from 1 to 0. However, because of the delay, it remains 1 for a period of time. During that time bits D0 through D5 are all 1's, producing the intensity value of 63 (white). The time sequence which produces this transition is shown in Figure 10.

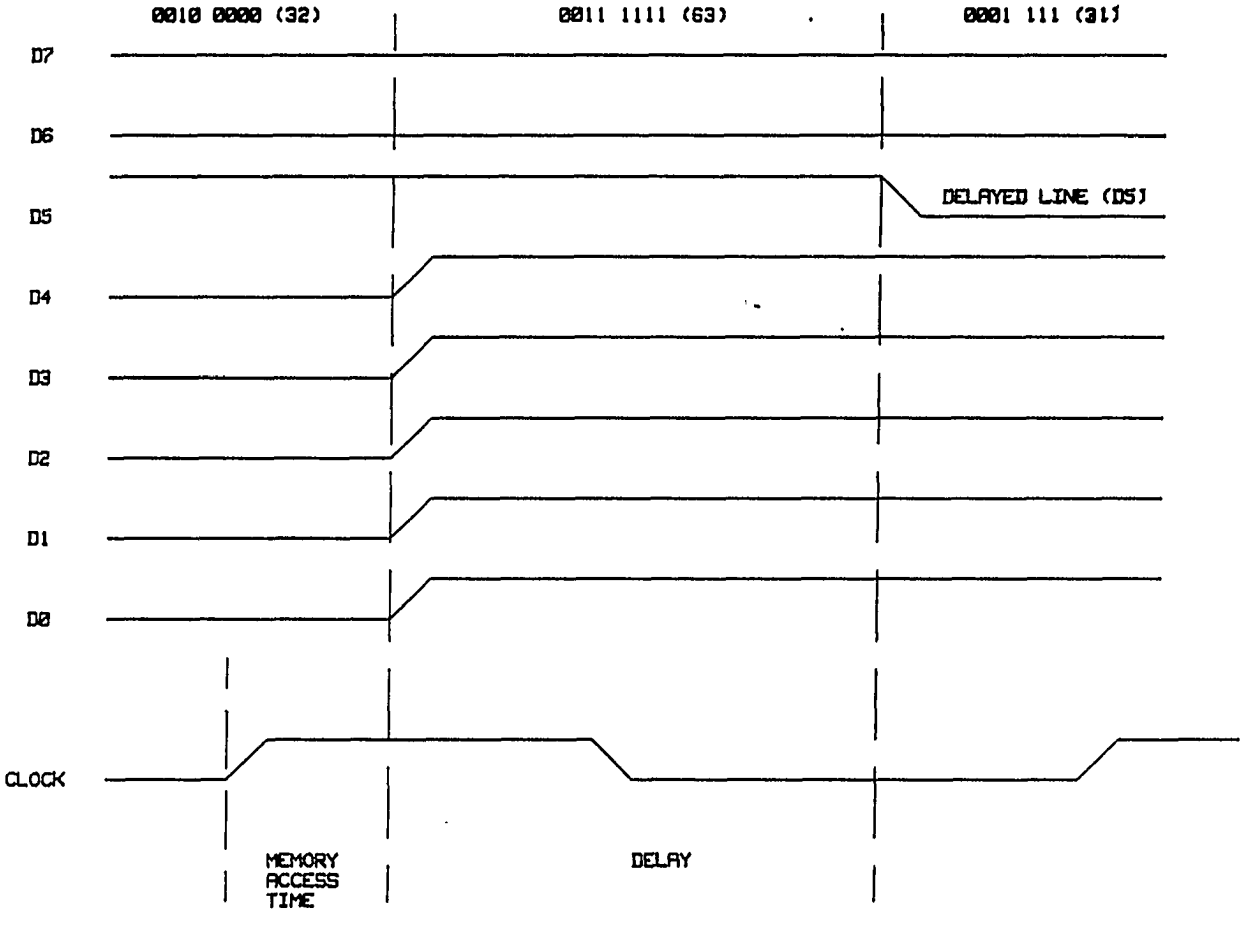


Figure 10. Timing diagram for white spot

VI. EXPERIMENTAL RESULTS

This section will describe the changes introduced into the original picture by varying the timing delay in the data lines of the Look-Up table circuit.

The most interesting effects on the edges appearing in a digitized image were observed by changing the timing characteristics of data lines D4 and D5, independently.

It was stated in a previous section that five different phases of interest could be identified. These phases were then photographed and they now show changes that are of interest in the edge detection variations.

It was observed that of the values of capacitors that were produced, a noticeable effect on the edges of the image were 0.001μf, 0.002μf, 0.003μf, and 0.004μf. Below or above these values, there was very little or no effect on the edge detection scheme of the image.

We will now described the different photographs that were taken of an equal number of images. The photographs are divided into two groups: two-dimensional images and three-dimensional images. It was learned that timing changes on only data lines 5 (D4) and 6 (D5) affected the edges on the image, as can be seen on all photographs.

A. Two-Dimensional Images

The first four photographs show the effects of changing the timing on data line D4 and the next four photographs show the effects of changing the timing on data line D5. In all cases, the time delay introduced on those lines was the same as in Photographs 9-13.

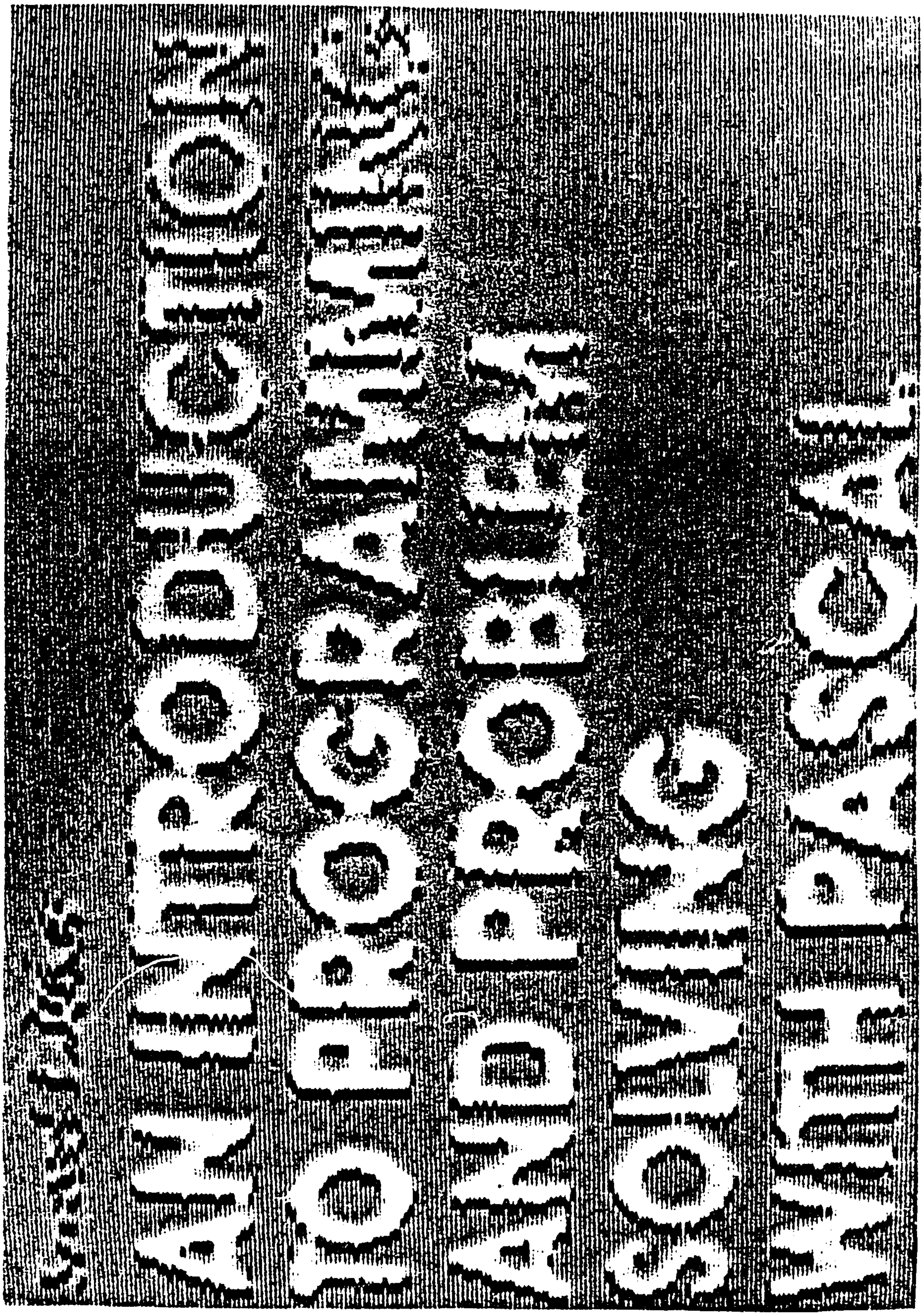
1. Photograph 14

Black contours appear along those edges of the letters that consist of a transition of low to high values as the screen is scanned from left to right. The contours are thin, their width being probably one pixel. They are generally continuous, having small gaps only when two letters practically touch each other, thus not allowing any low-valued background in between. The small letters in the upper left corner are not readily distinguishable.

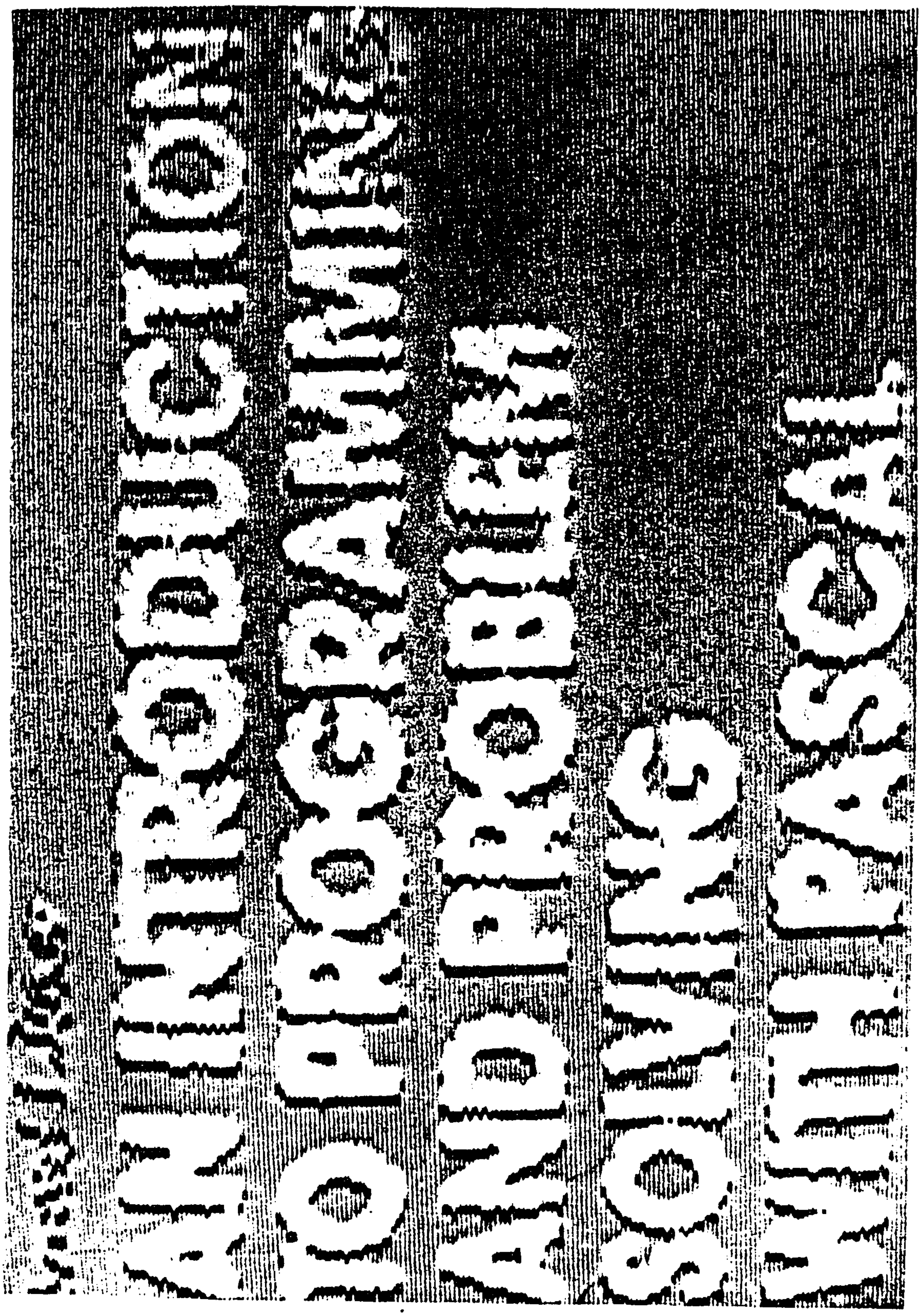
2. Photograph 15

In addition to the black contours, white contours now appear along the opposite transition (high to low values) as the screen is scanned from left to right. These contours follow the edged closely, but contain more gaps than the black ones. Some white dots also begin to appear around the small letters at the upper left corner.

Photograph 14. View of image with 0.001 μF capacitance on D4


AN INTRODUCTION
TO PROGRAMMING
AND PROBLEM
SOLVING
WITH PASCAL

Photograph 15. View of image with 0.002 μF capacitance on D4


AN INTRODUCTION
TO PROGRAMMING
AND PROBLEM
SOLVING
WITH PASCAL

3. Photograph 16

The black contours have doubled in width along some edges, while the white contours have become thinner.

4. Photograph 17

More black contours have doubled in width, while the white contours remain as in Photograph 16.

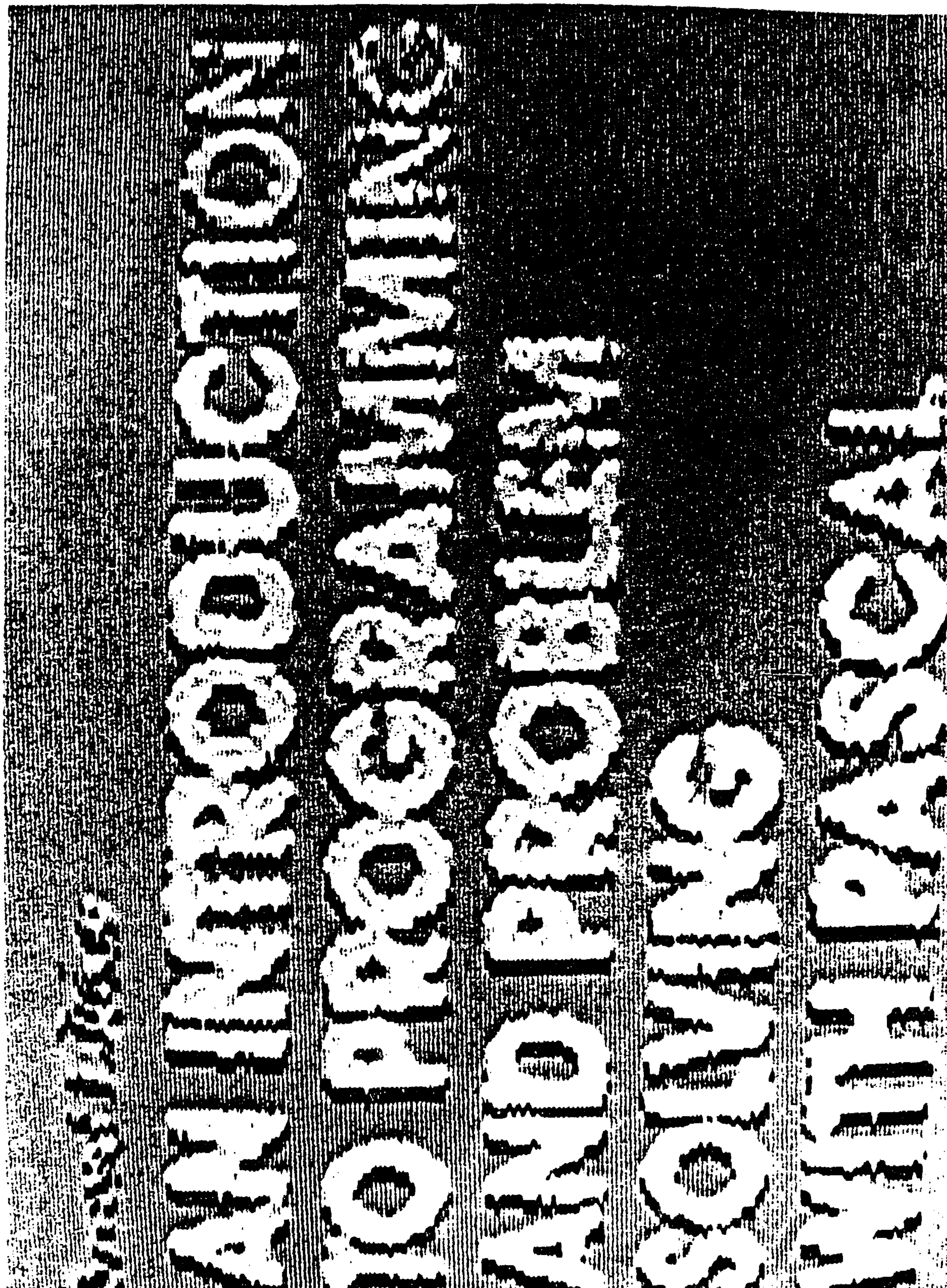
5. Photograph 18

Black contours appear along those edges of the letters that consist of a transition of high to low values as the screen is scanned from left to right. The contours are thin. Their width is probably one pixel. They are continuous, having small gaps only when two letters are so close so as to practically touch each other and thus do not allow any low-valued background in between.

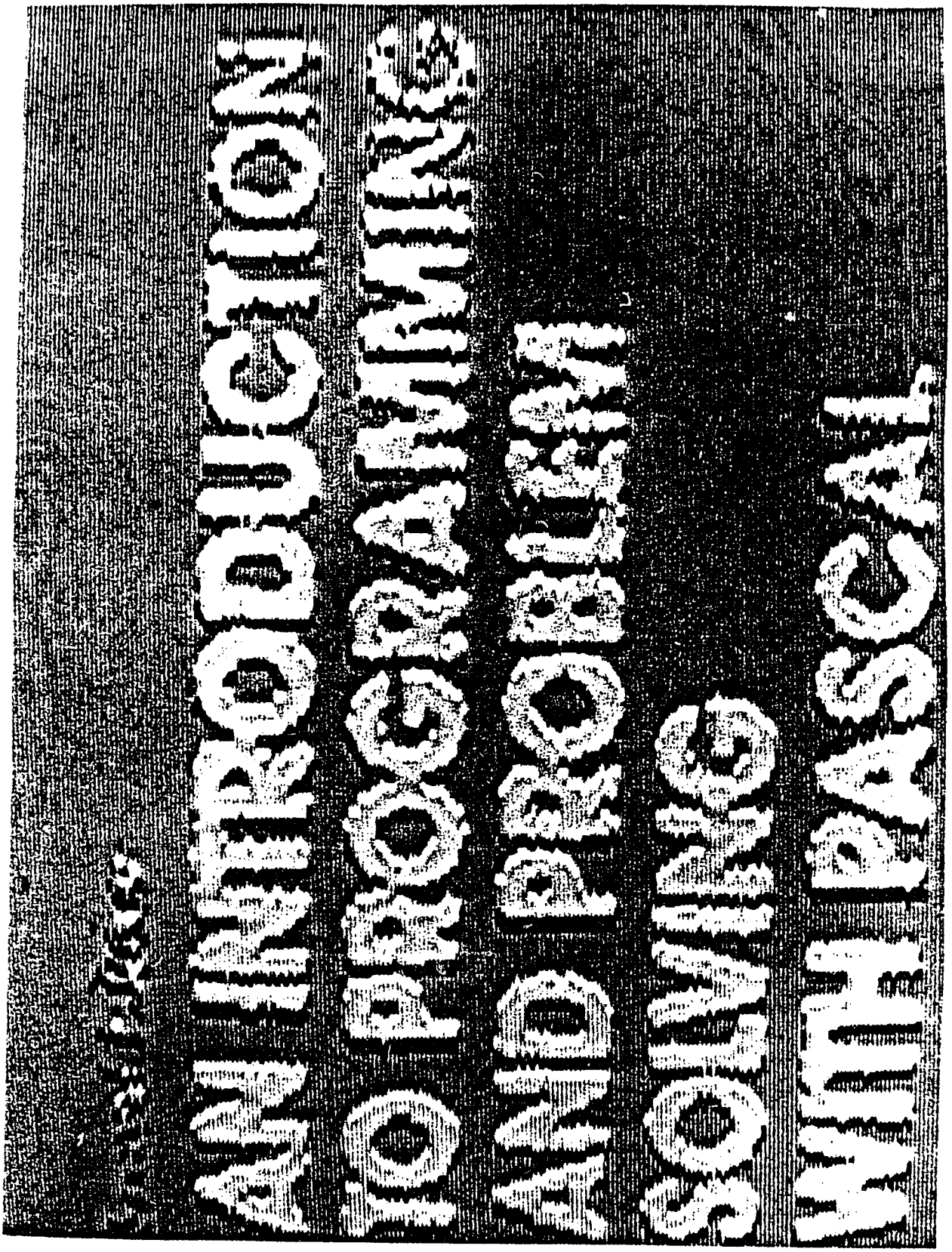
6. Photograph 19

White contours appear long the opposite transitions (low to high values), while the black contours remain unchanged. The new contours follow quite closely the edges of the letters. A pronounced three-dimensional effect may be observed.

Photograph 16. View of image with 0.003 μF capacitance on D4

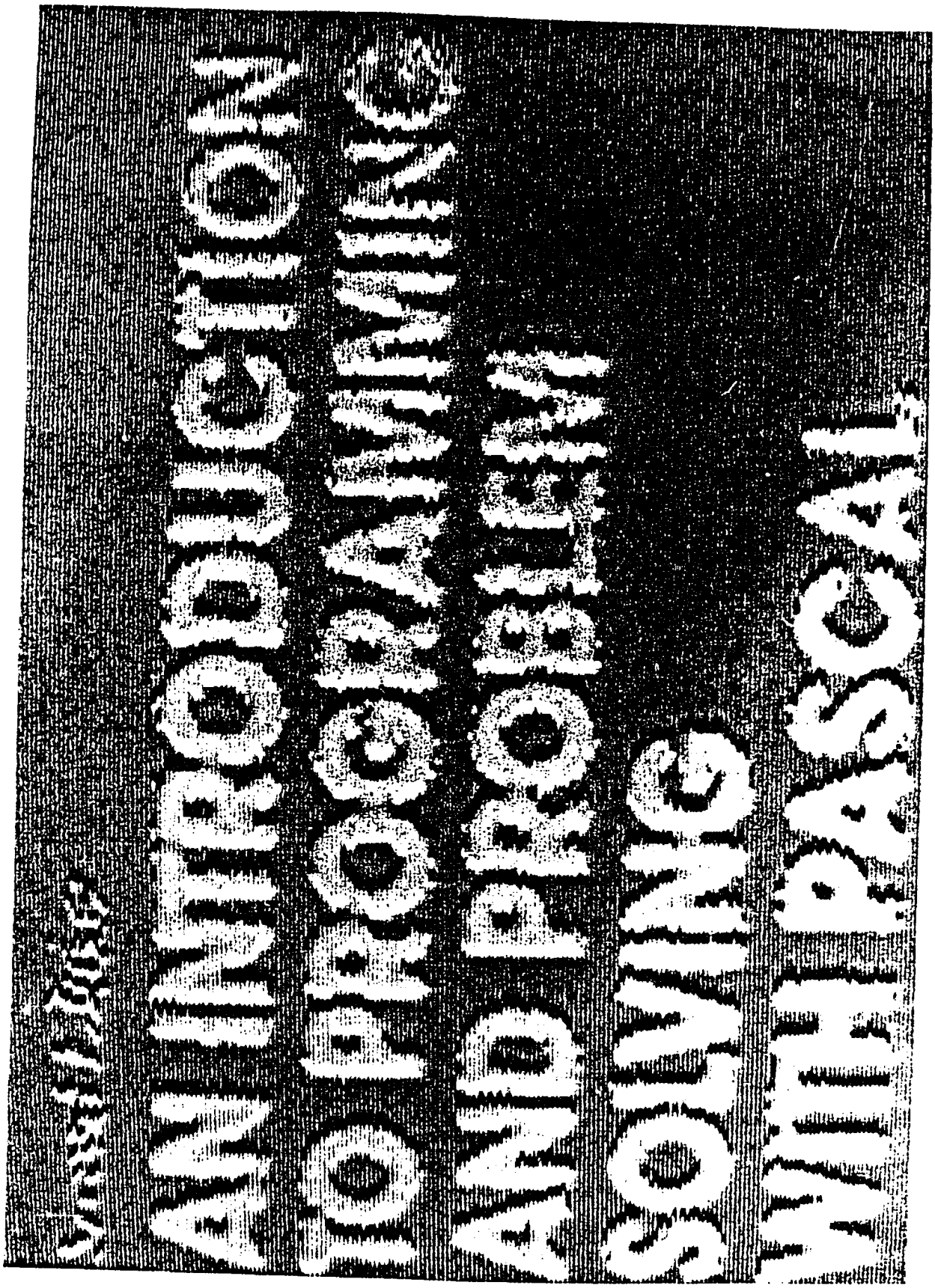

AN INTRODUCTION
TO PROGRAMMING
AND PROBLEM
SOLVING
WITH PASCAL

Photograph 17. View of image with 0.004 μF capacitance on D4


PRODUCTION
PRODUCTION

Photograph 18. View of image with 0.001 μF capacitance on D5

Photograph 19. View of image with 0.002 μF capacitance on D5



7. Photograph 20

Most of the black contours have doubled in width, while some parts of the white contours have disappeared. This appears to be an artifact of the film process.

8. Photograph 21

The black contours have expanded to almost three times their original width. The white ones remain as in Photograph 15. This also is an artifact.

B. Three-Dimensional Images

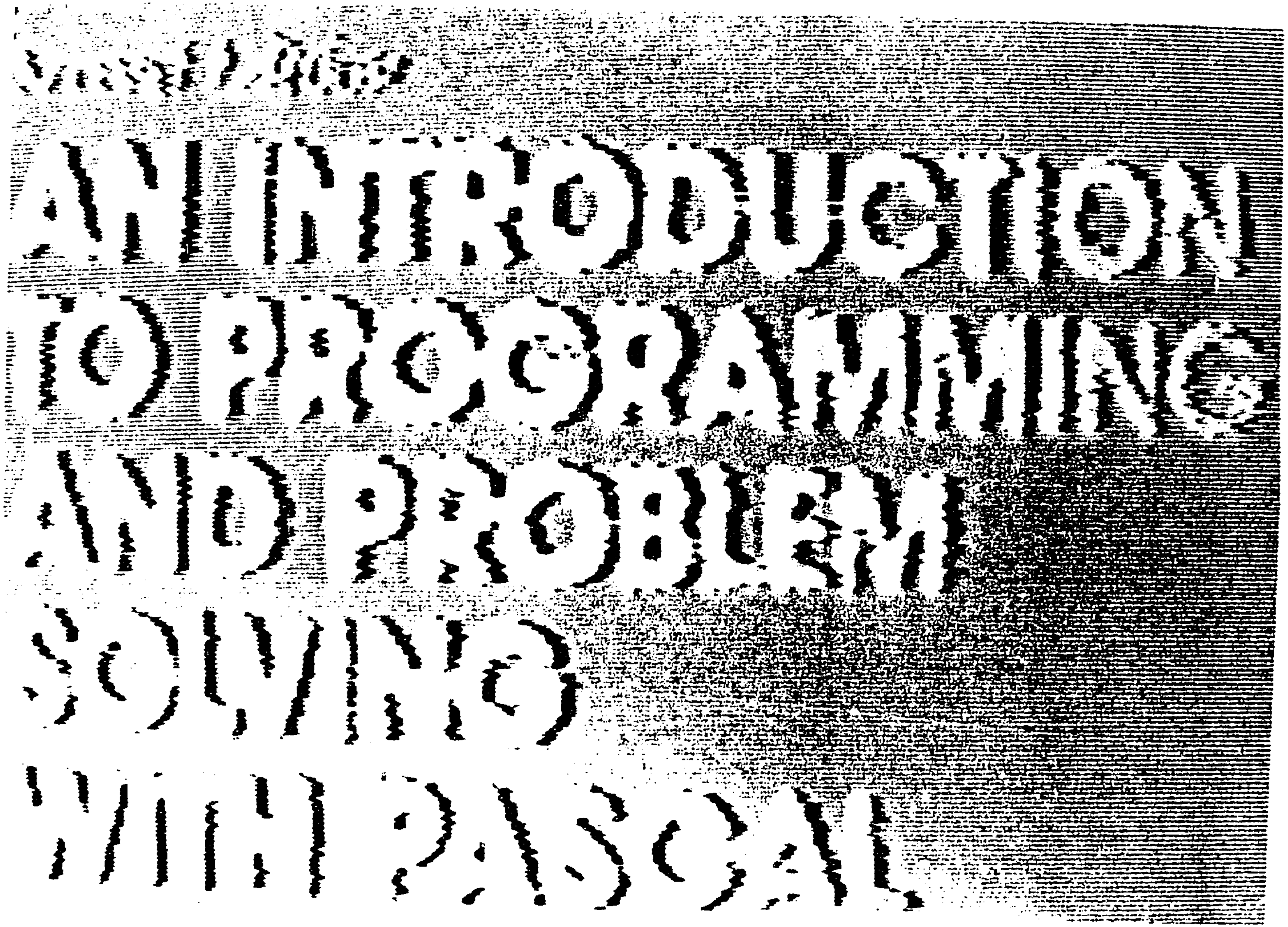
1. Photograph 22

Original sonographic image of a heart. When displayed, the image is inversed, i.e., the black becomes white and vice versa for demonstration reasons.

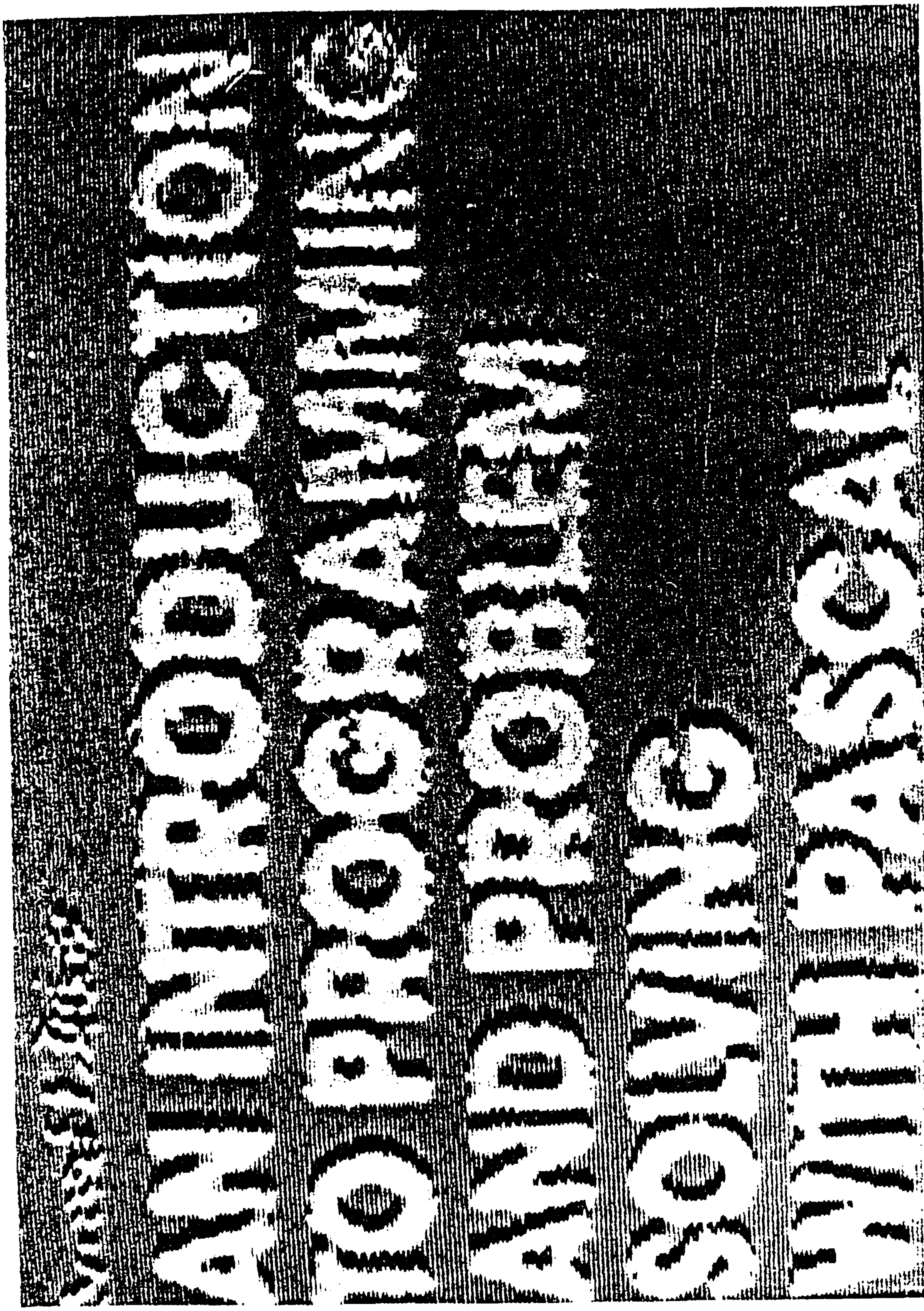
2. Photograph 23

Dark (almost black) contours appear along some transitions from high to low values as the image is scanned from left to right, while gray contours appear along some transitions from low to high values. The contours are not always continuous and seem to appear along specific transition of values all over the image. Their value is proportional to the steepness of the edges, i.e., the dark contours appear along greater transitions than the gray contours.

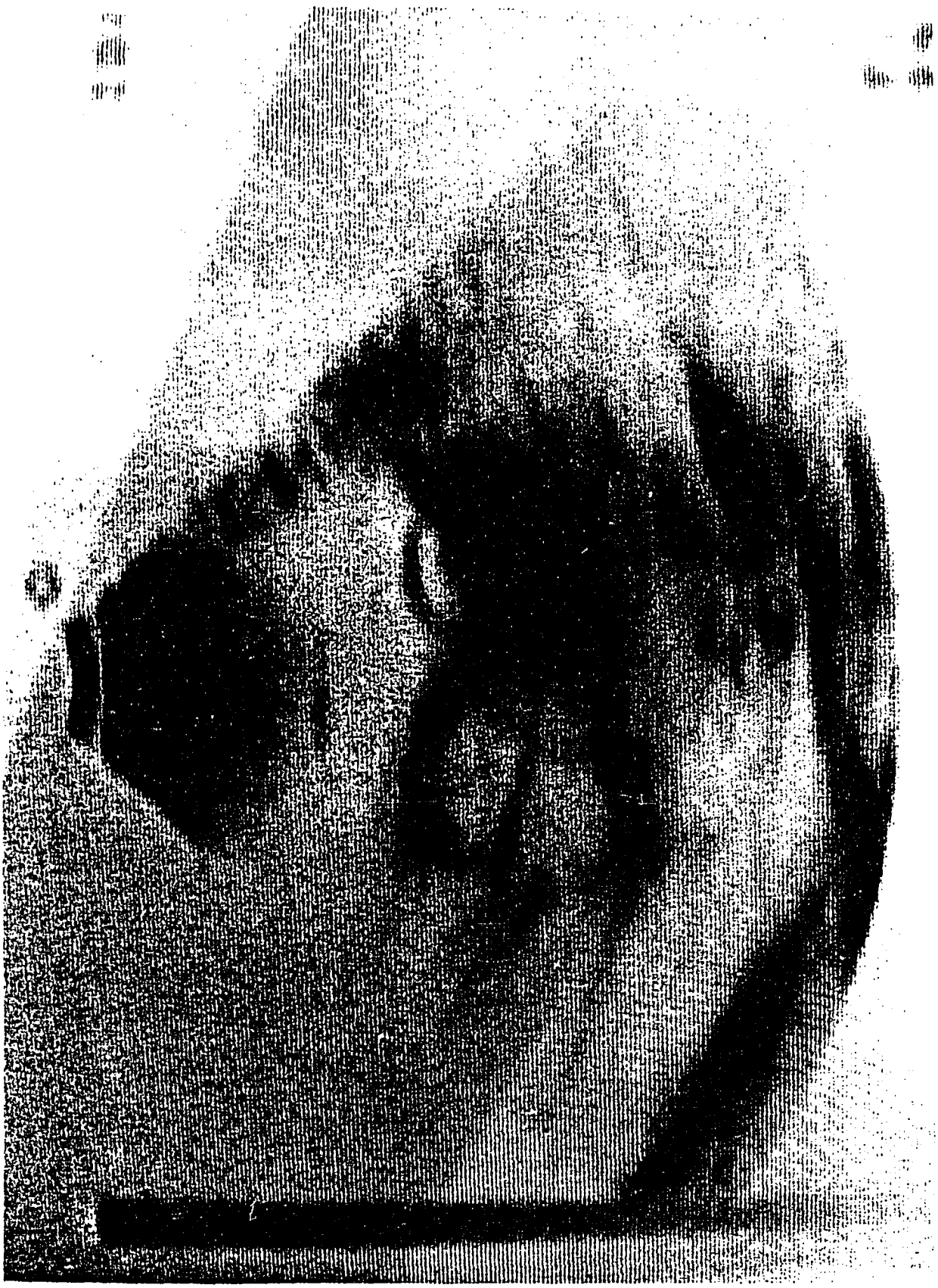
Photograph 20. View of image with 0.003 μF capacitance on D5


AN INTRODUCTION
TO PROGRAMMING
AND PROBLEM
SOLVING
WITH PASCAL

Photograph 21. View of image with 0.004 μF capacitance on D5


AN INTRODUCTION
TO PROGRAMMING
AND PROBLEM
SOLVING
WITH PASCAL

Photograph 22. Original image of heart



Photograph 23. View of heart's image with 0.001 μF

3. Photograph 24

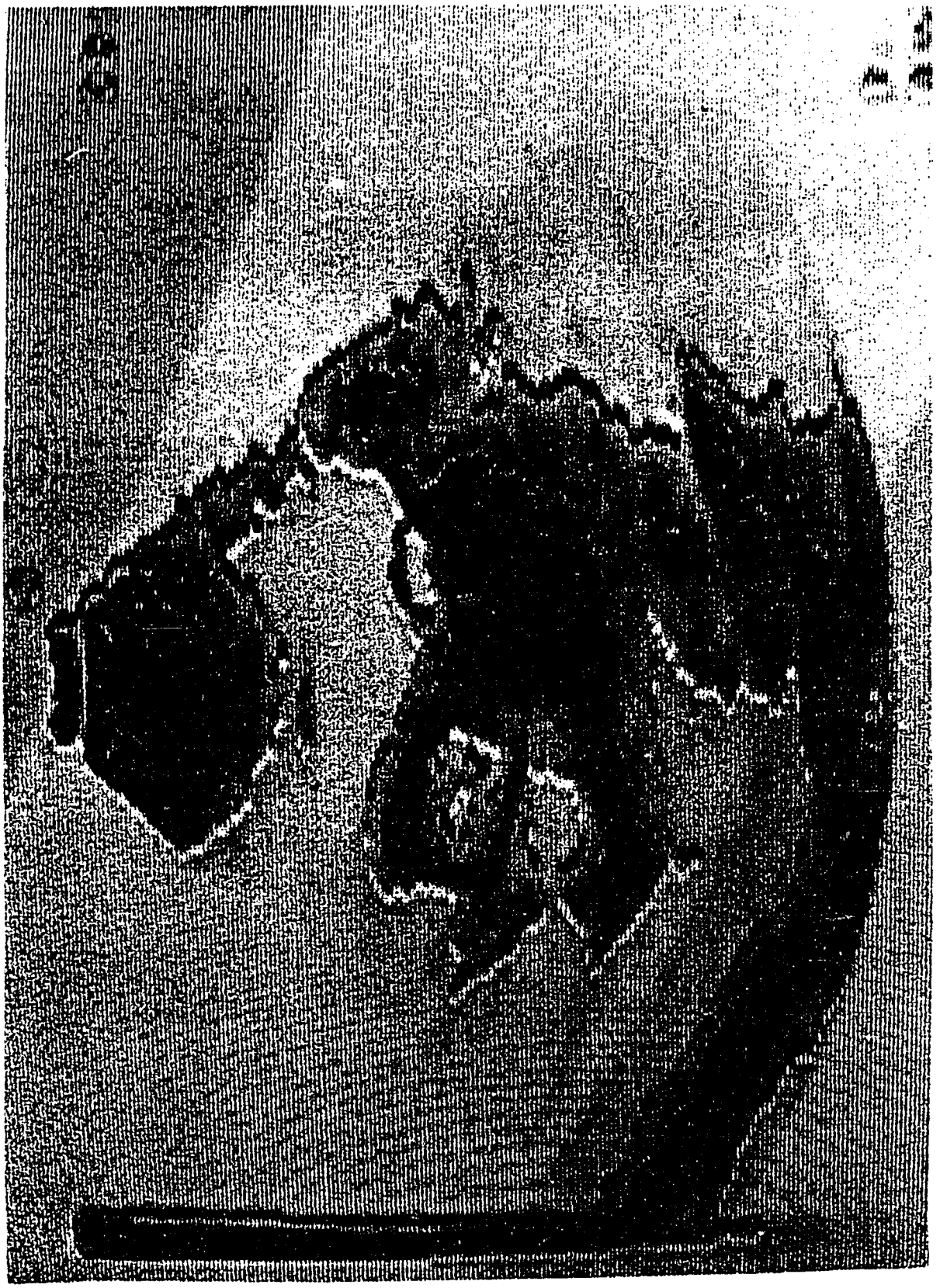
Some of the dark contours have doubled in width. Most of them, however, remain as in Photograph 23, while new ones, significantly brighter, appear along transitions from high to low values.

Less bright contours appear along some opposite transitions from low to high values. Again, the new contours seem to appear along specific transitions. Together with the old ones, they form almost closed boundaries of regions that have pixels with similar gray levels, creating the impression that the image shown is three-dimensional.

4. Photograph 25

Black contours appear along abrupt transitions from low to high values as the image is scanned from left to right. Their width is not uniform, nor are they everywhere continuous. Most of the gaps appear along horizontal edges as a result of the way the image is scanned. Close examination of Photographs 24 and 25 together reveals that the black contours of the latter appear at exactly the same position as the less bright of the white contours of the former.

Photograph 24. View of heart's image with 0.002 μF



Photograph 25. View of heart's image with 0.003 μF

5. Photograph 26

The black contours remain almost unchanged, while white contours appear along the opposite transitions from high to low values. They are more prominent wherever the edge is verticl and fail to appear wherever the edge is horizontal. Close examination of Photographs 24 and 26 or 23 reveals that the white contours of the former appear at exactly the same places as the darker ones of the latter.

6. Photograph 27

Original tomographic image of the head.

7. Photograph 28

Black contours and isolated black pixels appear along some transitions from high to low values as the image is scanned from left to right. The bigger contours appear at the right side of the image along the brain and along the outer side of the head.

8. Photograph 29

Some of the black contours have disappeared, but most of them have doubled and sometimes tripled in width. New midgray contours have appeared along transitions from low to high values, provided that the concerned edge is vertical. The most prominent mid-gray contour appears along the right side of the image.

Photograph 26. View of heart's image with 0.004 μF

Photograph 27. Original image of brain

Photograph 28. View of brain's image with 0.001 μF

Photograph 29. View of brain's image with 0.002 μF

9. Photograph 30

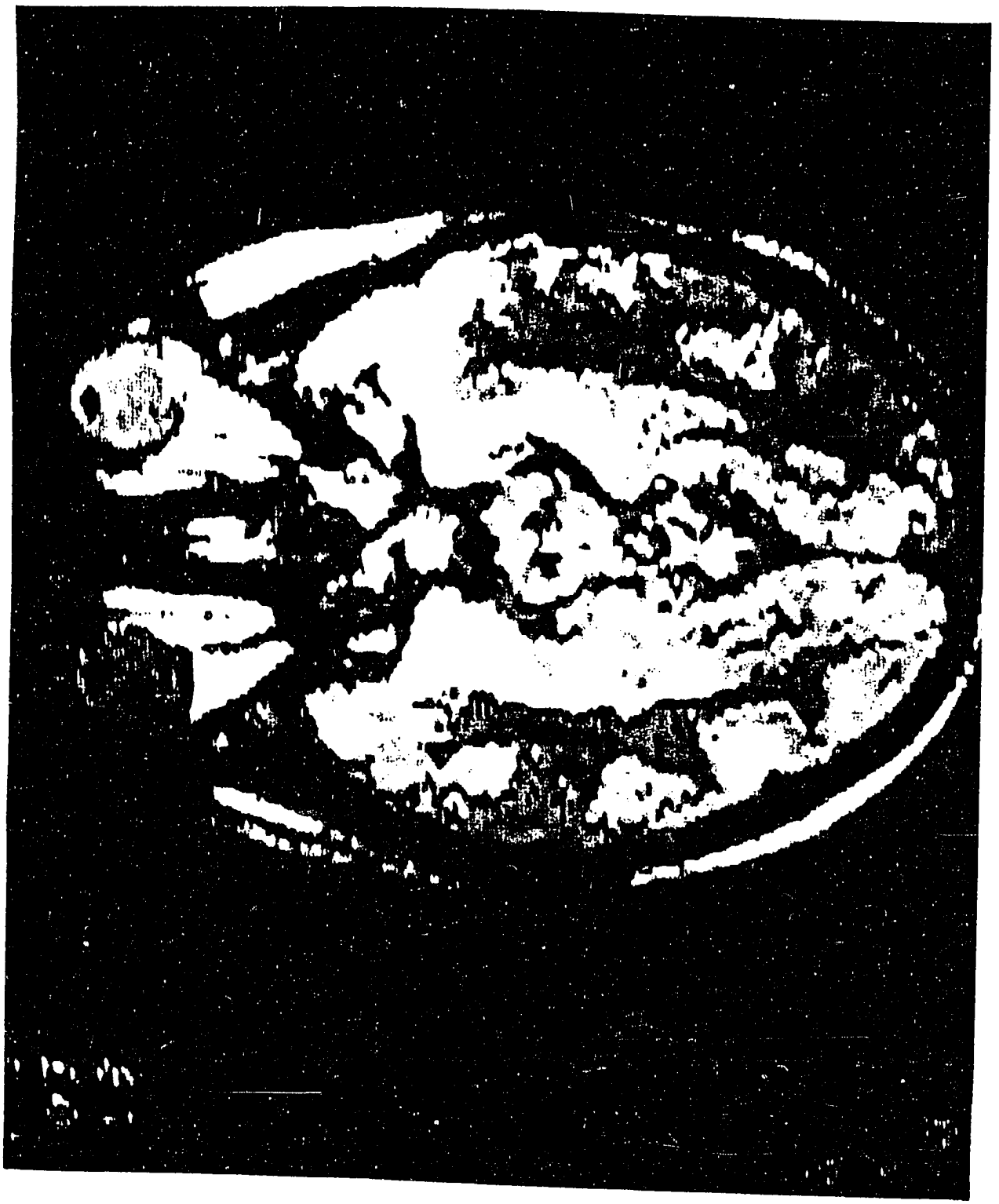
The contours have been inverted. Now black contours appear along transitions from low to high values, and gray ones, significantly brighter, along the opposite transitions. The contours do not appear exactly at the same place as in Photograph 29, but generally around the same regions. Most of them have uniform width.

10. Photograph 31

The contrast of the image has been drastically enhanced. Black contours appear along transitions from low to high values, and white contours along the opposite transitions. These contours have nothing to do with those of Photograph 30. They appear at totally different regions which have been separated from their neighboring regions due to the contrast enhancement introduced. For example, no contours now appear at the left eye, while new big contours have appeared at the left lobe of the brain around the dark region.

Photograph 30. View of brain's image with 0.003 μF

Photograph 31. View of brain's image with 0.004 μF



VII. CONCLUSIONS

A simple but powerful hardware edge detection method was developed and implemented as part of an Image Processing System. Its performance was tested against and compared to a number of conventional edge detection methods in a two-dimensional environment. Extending the performance test, three-dimensional medical images were used to evaluate the effectiveness of this method.

The performance of the method was satisfactory, both in two-dimensional and three-dimensional images. The performance advantage of the method in the two-dimensional environment shown in the photographs can be summarized as follows:

1. The exceptionally high speed with which this method gives results (in the order of milliseconds versus minutes as in the conventional Edge Detection Methods) makes this method very suitable for real-time edge detection applications that require human interactivity. None of the other conventional methods can qualify for such interactive edge detection applications in real time.
2. The lack of sensitivity this method displays in the presence of noise makes it very attractive to a large number of images in which more conventional methods need extensive processing to cope with

noise.

3. The hardware implementation method could have potential advantages over conventional methods with respect to danger of corruption, possible bugs, inefficient performance with different image characteristics, etc.
4. The cost for implementing this method as a hardware extension of any frame grabber is extremely low when compared with the cost of software edge detection methods. This can be translated into terms of processing speed and memory requirements. The flexibility, versatility, and simplicity of this method make it very attractive to perform an operation of detecting edges in raster computer images.

It should be noted that the results of the Hardware Edge Detection method in the three-dimensional environment (was represented by the medical images) were judged very satisfactory, by medical doctors Chris Venetis [14] and John Laskaris [15] after their initial observations of the results.

Currently, three different research projects are underway with the goal of detecting objects in medical images using this method. The results of these projects will be published soon.

Although the proposed High Speed Edge Detection method may outperform more conventional methods in achieving human interactivity, good performance in the presence of noise, and potentially higher reliability, it is clear that further work can be done in order to quantify the performance of this method and identify more specifically the areas in which it can be used.

In carrying out this project, the information contained in a number of handbooks and databooks [16-25] was very useful.

VIII. BIBLIOGRAPHY

1. Gonzalez, D. R. and Wintz P. "Digital Image Processing." Addison/Wesley Publishing Company, Reading, Mass., 1987.
2. Niblack, W. "An Introduction to Digital Image Processing." Prentice-Hall International, Englewood Cliffs, New Jersey, 1986.
3. Prewitt, W. K. "Digital Image Processing." J. Wiley, New York, 1978.
4. Castelman, K. R. "Digital Image Processing." Prentice-Hall, Englewood Cliffs, New Jersey, 1979.
5. Ekstrom, M. P. "Digital Image Processing Techniques." Academic Press, New York, 1984.
6. Ballard, D. H. and Brown, C. M. "Computer Vision." Prentice-Hall, Englewood Cliffs, New Jersey, 1982.
7. Levine, M. D. "Vision in Man and Machine." McGraw Hill, New York, 1985.
8. Venetsanopoulos, A. "Signal Processing." North Holland, The Hague, 1987.
9. Herskovits, A. and Binford, T. O. "On Boundary Detection," MIT Project MS4AC, Artificial Intelligence Memo 183, July 1970.

10. Fram, J. R. and Deutsch, E. S. "On the Evaluation of Edge Detection Schemes and Their Comparison with Human Performance." IEEE Trans. Computers, C-24, No. 6 (June 1975):616-628.

11. Schneider, G. M., Weingart, W. S., and Perlman, M. D. "An Introduction to Programming and Problem-Solving with Pascal." John Wiley & Sons, New York, 1982.

12. Rosenfeld, A. "Digital Image Processing Techniques." Academic Press, New York, 1987.

13. Venetsanopoulos, A. "Digital Image Processing and Analysis." University of Toronto, Ontario, 1985.

14. Venetis, Chris, M. D., Heart Specialist, Director of the Heart Clinic, Athens Naval Hospital, Athens, Greece. (Personal conversation).

15. Laskaris, John, M. D., President of the Greek Echography Society, Athens, Greece. (Personal conversation).

16. "Motorola Linear Interface Integrated Circuits." Motorola Semiconductor Products Inc., Phoenix, Arizona, 1979.

17. "Linear Data Book." NS Part No. IM-RRD85M31. National Semiconductor Corp., 2900 Semiconductor Drive, Santa Clara, California, 1980.

18. "The TTL Data Book for Design Engineers." TI Part No. LCC4112. Texas Instruments Inc., Semiconductor Group, Dallas, Texas, 1976.

19. "Microcontroller Handbook." Intel, 3065 Bowers Ave., Santa Clara, California, 1985.

20. "Microsystem Components Handbook." Volume I. Intel, 3065 Bowers Ave., Santa Clara, California, 1985.

21. "Memory Components Handbook." Intel, 3065 Bowers Ave., Santa Clara, California, 1984.

22. "Linear Applications Databook." National Semiconductor Corp., 2900 Semiconductor Drive, Santa Clara, California, 1978.

23. "Discrete Databook." National Semiconductor Corp., 2900 Semiconductor Drive, Santa Clara, California, 1978.

24. "Logic Databook Volume I." National Semiconductor Corp., 2900 Semiconductor Drive, Santa Clara, California, 1984.

25. "Logic Databook Volume II." National Semiconductor Corp., 2900 Semiconductor Drive, Santa Clara, California, 1984.

IX. ACKNOWLEDGMENTS

This is the space where the author has the opportunity to express his gratitude to people who directly or indirectly contributed to the success of this work.

First and foremost, I would like to express my deep appreciation to Prof. T. A. Smay for his help and above all his granite patience that I did not manage to exhaust. I remain forever indebted to him.

I would also like to thank Prof. R. M. Steward for his understanding and sensitivity he displayed when he dealt with the case as Chair of the Department's Graduate Program Committee. It will be an omission not to mention Prof. R. Zingg who was my first advisor, consultant, and friend.

Special thanks to my friend Gary Bridges for his moral support and last but not least I would like to thank E. Pelecanos, G. Mitakides, S. Tsanaktsis and M. Kentouri for their technical support.

X. APPENDIX: PROGRAM LISTINGS FOR THE FILTERS

PROCEDURE CONVOLVE

```
(*---------------------*)
(*   C O N V O L V E   *)
(*---------------------*)

Procedure convolve(Table_In,Table_Out : stuff_in_heap;
                   var h : C_Window; N,Scale : integer;
                   x1,y1,x2,y2 : integer);
var
   xx,yy                : integer;
   max_real,min_real    : real;


Function convolve_point(y,x,N,Scale  : integer) : byte;

(*---------------------*)
(*   CONVOLVE  POINT   *)
(*---------------------*)

var
   i                              : integer;
   xx, yy                         : integer;
   value                          : real;

begin
     value := 0;
     i     := 0;
     for yy := y-N to y+N do
       for xx := x-N to x+N do
       begin
         value := value + table_in.matrix[yy,xx] * h[i];
         i := i + 1;
       end;
     value := value/Scale;
     if value > peak then
     begin
       value := peak;
       overflows := overflows + 1;
     end;
     if value<0 then
     begin
        value:=abs(value);
        underflows:=underflows+1;
     end;
     convolve_point := round(value);
end; (* of convolve_point *)
```

```
begin  (* of convolve *)
     if x1 < min_hor+N then x1 := min_hor+N;
     if y1 < min_ver+N then y1 := min_ver+N;
     if x2 > max_hor-N then x2 := max_hor-N;
     if y2 > max_ver-N then y2 := max_ver-N;

     writeln('Convolution Started');
     overflows := 0;
     underflows:=0;
     for yy := y1 to y2 do
     begin
       for xx := x1 to x2 do
        table_out.matrix[yy,xx] :=
                          convolve_point(yy,xx,N,Scale);
       write('.');
     end;

     writeln;
     writeln('Convolution Ended.');
     writeln('Number of Overflows  = ',overflows);
     writeln('Number of Underflows = ',underflows);
end;   (* of convolve *)

Procedure convolve2(Table_In,Table_Out : stuff_in_heap;
                    var h              : C_Window;
                    N,Scale,threshold  : integer;
                    x1,y1,x2,y2        : integer);
```

PROCEDURE THRESHOLDED CONVOLUTION

```
(*-------------------------------------------------*)
(*   T H R E S H O L D E D   C O N V O L U T I O N  *)
(*-------------------------------------------------*)
(*                                                 *)
(* The pixel is not changed if the output convolution*)
(* does not exceed  the threshold value.           *)
(*-------------------------------------------------*)
var
   xx,yy                : integer;
   max_real,min_real    : real;


Function convolve_point2(y,x,N,Scale  : integer) : byte;
(*----------------------*)
(*   CONVOLVE  POINT 2 *)
(*----------------------*)

var
   i                                 : integer;
   xx, yy                            : integer;
   value                             : real;

begin
     value := 0;
     i     := 0;
     for yy := y-N to y+N do
       for xx := x-N to x+N do
       begin
         value := value + table_in.matrix[yy,xx] * h[i];
         i := i + 1;
       end;
     value := value/Scale;
     if value < threshold then
                              value:=table_in.matrix[y,x];
     if value > peak then
     begin
       value := peak;
       overflows := overflows + 1;
     end;
     if value<0 then
     begin
        value:=0;
        underflows:=underflows+1;
     end;
     convolve_point2 := round(value);
end; (* of convolve_point2 *)
```

```
begin  (* of convolve2 *)
     if x1 < min_hor+N then x1 := min_hor+N;
     if y1 < min_ver+N then y1 := min_ver+N;
     if x2 > max_hor-N then x2 := max_hor-N;
     if y2 > max_ver-N then y2 := max_ver-N;

     writeln('Convolution Started');
     overflows := 0;
     underflows:= 0;
     for yy := y1 to y2 do
     begin
       for xx := x1 to x2 do
        table_out.matrix[yy,xx] :=
                          convolve_point2(yy,xx,N,Scale);
       write('.');
     end;

     writeln;
     writeln('Convolution Ended.');
     writeln('Number of Overflows  = ',overflows);
     writeln('Number of Underflows = ',underflows);
end;  (* of convolve2 *)
```

PROCEDURE EDGE-MATCHING

```
(*-------------------------------------------------*)
(*            E D G E - M A T C H I N G            *)
(*-------------------------------------------------*)
(*                                                 *)
(*    It computes the convolution of the x1-x2,y1-y2 *)
(*  window of the image with 8 masks and chooses as  *)
(*  output the greatest result. The elements of the  *)
(*  masks are asked at the input. It is used mainly  *)
(*  for edge detection purposes with masks that de-  *)
(*  tect edges at different directions.              *)
(*-------------------------------------------------*)

var
   h1,h2,h3,h4,h5,h6,h7,h8:C_Window;
   i,m_radius,window_length,scale:integer;
   value,threshold : integer;
   v:Histo_type;
   yy,xx:integer;


Function convolve_point3(y,x,N :integer;
                         h     :C_window) : byte;

(*---------------------*)
(*   CONVOLVE  POINT3  *)
(*---------------------*)

var
   i                              : integer;
   xx, yy                         : integer;
   value                          : real;

begin
     value := 0;
     i     := 0;
     for yy := y-N to y+N do
       for xx := x-N to x+N do
       begin
         value := value + table_in.matrix[yy,xx] * h[i];
         i := i + 1;
       end;
     if value < 0 then
     begin
        value:=abs(value);
        underflows:=underflows+1;
     end;
```

```
      if value > peak then
      begin
        value := peak;
        overflows := overflows + 1;
      end;
      convolve_point3 := round(value);
end; (* of convolve_point3 *)

begin  (* Edge_matching *)

   write('Enter mask"s radius : ');
   readln(m_radius);
   writeln;
   write('Enter the threshold : ');
   readln(threshold);
   writeln;
   writeln('Enter the elements of the masks : ');
   window_length:=(2*m_radius+1)*(2*m_radius+1);
   for i:=0 to window_length-1 do
   readln(h1[i]);
   for i:=0 to window_length-1 do
   readln(h2[i]);
   for i:=0 to window_length-1 do
   readln(h3[i]);
   for i:=0 to window_length-1 do
   readln(h4[i]);
   for i:=0 to window_length-1 do
   readln(h5[i]);
   for i:=0 to window_length-1 do
   readln(h6[i]);
   for i:=0 to window_length-1 do
   readln(h7[i]);
   for i:=0 to window_length-1 do
   readln(h8[i]);
   write('Scaling constant : ');
   readln(scale);

     writeln('Convolution Started');
     overflows := 0;
     underflows:= 0;
     for yy := y1 to y2 do
     begin
       for xx := x1 to x2 do
       begin
          v[0]:=convolve_point3(yy,xx,m_radius,h1);
          v[1]:=convolve_point3(yy,xx,m_radius,h2);
          v[2]:=convolve_point3(yy,xx,m_radius,h3);
          v[3]:=convolve_point3(yy,xx,m_radius,h4);
          v[4]:=convolve_point3(yy,xx,m_radius,h5);
```

```
            v[5]:=convolve_point3(yy,xx,m_radius,h6);
            v[6]:=convolve_point3(yy,xx,m_radius,h7);
            v[7]:=convolve_point3(yy,xx,m_radius,h8);
            value:=(maximum(v,8))div(scale);
            if value <= threshold then value:=0;
            table_out.matrix[yy,xx]:=value;
          end;
          write('.');
        end;

        writeln;
        writeln('Convolution Ended.');
        writeln('Number of Overflows  = ',overflows);
        writeln('Number of Underflows = ',underflows);

    end; (* Edge_matching *)
```

PROCEDURE INTERACTIVE FILTER

```
(*-------------------------------------------------*)
(*        I N T E R A C T I V E - F I L T E R      *)
(*-------------------------------------------------*)
(*                                                 *)
(*   It computes the convolution of a wx1-wx2,wy1-wy2 *)
(* window of the image with a mask of any size up to  *)
(* 21*21 which is asked at the input. There are two   *)
(* possible options :                              *)
(*   The output is the above mentioned convolution    *)
(* regardless of its value or the output differs from *)
(* the input only at those points for which the convo-*)
(* lution result exeeds a given threshold.         *)
(*-------------------------------------------------*)

var
  window_length,m_radius,i:integer;
  scale:integer;
  h:C_window;
  choice,katofli:integer;
begin

  write('Enter mask"s radius : ');
  readln(m_radius);
  writeln;
  write('Scale constant : ');
  readln(scale);
  if m_radius>10 then
      begin
        m_radius:=10;
        writeln('The value has been rounded to 10');
        writeln;
        readln;readln;
      end;
  window_length:=(2*m_radius+1)*(2*m_radius+1);
  writeln('Enter mask"s values : ');
  for i:=0 to window_length-1 do
  readln(h[i]);
  writeln('1.Simple convolution');
  writeln('2.Thresholded convolution');
  write('Your choice : ');
  readln(choice);
  writeln;
```

```
   case choice of
      1: convolve(table_in,table_out,h,m_radius,
                                 scale,wx1,wy1,wx2,wy2);
      2: begin
            writeln('Enter the threshold');
            readln(katofli);
         convolve2(table_in,table_out,h,m_radius,
                              scale,katofli,wx1,wy1,wx2,wy2);
         end;
    end;

end;  (* of Interactive_filter *)
```

PROCEDURE EXTENDED-PREWITT

```
(*-------------------------------------------------------*)
(*          E X T E N D E D  - P R E W I T T             *)
(*-------------------------------------------------------*)
(*                                                       *)
(*   It performs edge detection along 4 directions by    *)
(* means of local mean values next to the current point  *)
(* at a direction perpendicular to the one detected.     *)
(* The result is compared to a threshold and if it is    *)
(* greater, it is multiplied by a chosen constant (en-   *)
(* force) and it is given at the output. Otherwise the   *)
(* output is set equal to zero (background). The masks   *)
(* asked at the input should be Prewitt or Sobel or      *)
(* equivalent masks.                                     *)
(*-------------------------------------------------------*)

var
   i,j,k1,k2,k3,l1,l2,l3,m1,m2,m3,n1,n2,n3: integer;
   threshold,edge,enforce: integer;
   m_radius,window_length: integer;
   h1,h2,h3,h4: C_Window;

Function convolve_point(h:C_window; y,x,N,Scale  : integer) :
byte;
(*---------------------*)
(*   CONVOLVE  POINT   *)
(*---------------------*)

var
   i                              : integer;
   xx, yy                         : integer;
   value                          : real;
begin
     value := 0;
     i     := 0;
     for yy := y-N to y+N do
       for xx := x-N to x+N do
       begin
         value := value + table_in.matrix[yy,xx] * h[i];
         i := i + 1;
       end;
     value := value/Scale;
     if value > peak then
     begin
       value := peak;
       overflows := overflows + 1;
     end;
```

```
      if value<0 then
      begin
         value:=abs(value);
         underflows:=underflows+1;
      end;
      convolve_point := round(value);
end; (* of convolve_point *)

function max(a1,a2,a3,a4:integer):integer;
var maxi :integer;
begin
  maxi:=a1;
  if maxi<a2 then maxi:=a2;
  if maxi<a3 then maxi:=a3;
  if maxi<a4 then maxi:=a4;
  max:=maxi;
end;

begin
write('Enter the mask"s radius : ');
readln(m_radius);
writeln;
window_length:=(2*m_radius+1)*(2*m_radius+1);
write('Enter the threshold value : ');
readln(threshold);
writeln;
write('Enter the enforcement factor : ');
readln(enforce);
writeln;
writeln('Enter the values of the masks');
for i:=0 to window_length-1 do
readln(h1[i]);
for i:=0 to window_length-1 do
readln(h2[i]);
for i:=0 to window_length-1 do
readln(h3[i]);
for i:=0 to window_length-1 do
readln(h4[i]);

for i:=wy1 to wy2 do
begin
  for j:=wx1 to wx2 do
  begin
    k1:=abs(convolve_point(h1,i,j-1,m_radius,1));
    k2:=abs(convolve_point(h1,i,j,m_radius,1));
    k3:=abs(convolve_point(h1,i,j+1,m_radius,1));
    if (k2<k1) or (k2<k3) then k2:=0;
```

```
      m1:=abs(convolve_point(h2,i-1,j-1,m_radius,1));
      m2:=abs(convolve_point(h2,i,j,m_radius,1));
      m3:=abs(convolve_point(h2,i+1,j+1,m_radius,1));
      if (m2<m1) or (m2<m3) then m2:=0;
      l1:=abs(convolve_point(h3,i-1,j,m_radius,1));
      l2:=abs(convolve_point(h3,i,j,m_radius,1));
      l3:=abs(convolve_point(h3,i+1,j,m_radius,1));
      if (l2<l1) or (l2<l3) then l2:=0;
      n1:=abs(convolve_point(h4,i-1,j+1,m_radius,1));
      n2:=abs(convolve_point(h4,i,j,m_radius,1));
      n3:=abs(convolve_point(h4,i+1,j-1,m_radius,1));
      if (n2<n1) or (n2<n3) then n2:=0;
      edge:=max(k2,m2,l2,n2);

      if edge>threshold
         then table_out.matrix[i,j]:=enforce*edge
         else table_out.matrix[i,j]:=0;
    end;
    write('.');
  end;

    writeln;
    writeln('Number of overflows : ',overflows);
    writeln('Number of underflows : ',underflows);
  end; (* of Extended Prewitt *)
```

PROCEDURE PREWITT-2

```
(*-------------------------------------------------*)
(*              P R E W I T T - 2                  *)
(*-------------------------------------------------*)
(*                                                 *)
(*   It performs edge detection along the horizontal *)
(* and vertical direction using local mean values at *)
(* areas next to the current point. The process is   *)
(* carried out at a wx1-wx2,wy1-wy2 window of the    *)
(* image and the result is not thresholded.          *)
(*-------------------------------------------------*)

var
  i,j,k1,k2,k3,l1,l2,l3:integer;
  ti:stuff_in_heap;

begin  (* Prewitt2 *)
   ti:=table_in;
   for i:=wy1 to wy2 do
   begin
     for j:=wx1 to wx2 do
     begin
        k1:=abs(ti.matrix[i-1,j] + ti.matrix[i,j]
              + ti.matrix[i+1,j] - ti.matrix[i-1,j-2]
              - ti.matrix[i,j-2] - ti.matrix[i+1,j-2]);
         k2:=abs(ti.matrix[i-1,j+1] + ti.matrix[i,j+1]
               + ti.matrix[i+1,j+1] - ti.matrix[i-1,j-1]
               - ti.matrix[i,j-1] - ti.matrix[i+1,j-1]);
        k3:=abs(ti.matrix[i-1,j+2] + ti.matrix[i,j+2]
              + ti.matrix[i+1,j+2] - ti.matrix[i-1,j]
              - ti.matrix[i,j] - ti.matrix[i+1,j]);
        if (k2<k1) or (k2<k3) then k2:=0;

        l1:=abs(ti.matrix[i-2,j-1] + ti.matrix[i-2,j]
              + ti.matrix[i-2,j+1] - ti.matrix[i,j-1]
              - ti.matrix[i,j] - ti.matrix[i,j+1]);
         l2:=abs(ti.matrix[i-1,j-1] + ti.matrix[i-1,j]
               + ti.matrix[i-1,j+1] - ti.matrix[i+1,j-1]
               - ti.matrix[i+1,j] - ti.matrix[i+1,j+1]);
         l3:=abs(ti.matrix[i,j+1] + ti.matrix[i,j]
               + ti.matrix[i,j-1] - ti.matrix[i+2,j-1]
               - ti.matrix[i+2,j] - ti.matrix[i+2,j+1]);
         if (l2<l1) or (l2<l3) then  l2:=0;
```

```
        if k2>12 then
                if k2>peak  then
                                table_out.matrix[i,j]:=peak
                else
                    table_out.matrix[i,j]:=k2
        else
                 if 12>peak  then
                                 table_out.matrix[i,j]:=peak
                 else
                     table_out.matrix[i,j]:= 12;
    end;
end;
end;   (* Prewitt2 *)
```

PROCEDURE PREWITT-2A

```
(*-------------------------------------------------*)
(*                P R E W I T T - 2A               *)
(*-------------------------------------------------*)
(*                                                 *)
(*   Same as Procedure Prewitt2 the only difference  *)
(* being a threshold which determines which points   *)
(* will be amplified (multiplied by 3) so that the   *)
(* edges be more prominent.                          *)
(*-------------------------------------------------*)

var
  i,j,k1,k2,k3,11,12,13:integer;
  ti:stuff_in_heap;
  threshold:integer;

begin  (* Prewitt2a *)
   write('Enter the value of the threshold : ');
   readln(threshold);
   writeln;
   ti:=table_in;
   for i:=wy1 to wy2 do
   begin
     for j:=wx1 to wx2 do
     begin
        k1:=abs(ti.matrix[i-1,j] + ti.matrix[i,j]
              + ti.matrix[i+1,j] - ti.matrix[i-1,j-2]
              - ti.matrix[i,j-2] - ti.matrix[i+1,j-2]);
         k2:=abs(ti.matrix[i-1,j+1] + ti.matrix[i,j+1]
               + ti.matrix[i+1,j+1] - ti.matrix[i-1,j-1]
               - ti.matrix[i,j-1] - ti.matrix[i+1,j-1]);
        k3:=abs(ti.matrix[i-1,j+2] + ti.matrix[i,j+2]
              + ti.matrix[i+1,j+2] - ti.matrix[i-1,j]
              - ti.matrix[i,j] - ti.matrix[i+1,j]);
        if (k2<k1) or (k2<k3) then k2:=0;

         11:=abs(ti.matrix[i-2,j-1] + ti.matrix[i-2,j]
               + ti.matrix[i-2,j+1] - ti.matrix[i,j-1]
               - ti.matrix[i,j] - ti.matrix[i,j+1]);
         12:=abs(ti.matrix[i-1,j-1] + ti.matrix[i-1,j]
               + ti.matrix[i-1,j+1] - ti.matrix[i+1,j-1]
               - ti.matrix[i+1,j] - ti.matrix[i+1,j+1]);
         13:=abs(ti.matrix[i,j+1] + ti.matrix[i,j]
               + ti.matrix[i,j-1] - ti.matrix[i+2,j-1]
               - ti.matrix[i+2,j] - ti.matrix[i+2,j+1]);
```

```
        if (l2<l1) or (l2<l3) then  l2:=0;
        if k2>=threshold then k2:=3*k2;
        if l2>=threshold then l2:=3*l2;
        if k2>l2 then
                if k2>peak  then
                            table_out.matrix[i,j]:=peak
                else
                    table_out.matrix[i,j]:=k2
        else
                 if l2>peak  then
                             table_out.matrix[i,j]:=peak
                 else
                     table_out.matrix[i,j]:= l2;
    end;
end;
end;   (* Prewitt2a *)
```

PROCEDURE SOBEL-2A

```
(*-------------------------------------------------------*)
(*                    S O B E L - 2A                     *)
(*-------------------------------------------------------*)
(*                                                       *)
(*   It is the same as Procedure Prewitt2a but it uses   *)
(* different weight coefficients for the calculation     *)
(* of the local mean values. The weight coefficients     *)
(* the same as in Procedure Sobel1.                      *)
(*-------------------------------------------------------*)

var
  i,j,k1,k2,k3,l1,l2,l3:integer;
  ti:stuff_in_heap;
  threshold:integer;

begin  (* Sobel2a *)
   write('Enter the value of the threshold : ');
   readln(threshold);
   writeln;
   ti:=table_in;
   for i:=wy1 to wy2 do
   begin
     for j:=wx1 to wx2 do
     begin
        k1:=abs(ti.matrix[i-1,j] + 2*ti.matrix[i,j]
             + ti.matrix[i+1,j] - ti.matrix[i-1,j-2]
             - 2*ti.matrix[i,j-2] - ti.matrix[i+1,j-2]);
         k2:=abs(ti.matrix[i-1,j+1] + 2*ti.matrix[i,j+1]
              + ti.matrix[i+1,j+1] - ti.matrix[i-1,j-1]
              - 2*ti.matrix[i,j-1] - ti.matrix[i+1,j-1]);
        k3:=abs(ti.matrix[i-1,j+2] + 2*ti.matrix[i,j+2]
             + ti.matrix[i+1,j+2] - ti.matrix[i-1,j]
             - 2*ti.matrix[i,j] - ti.matrix[i+1,j]);
        if (k2<k1) or (k2<k3) then k2:=0;

         l1:=abs(ti.matrix[i-2,j-1] + 2*ti.matrix[i-2,j]
              + ti.matrix[i-2,j+1] - ti.matrix[i,j-1]
              - 2*ti.matrix[i,j] - ti.matrix[i,j+1]);
         l2:=abs(ti.matrix[i-1,j-1] + 2*ti.matrix[i-1,j]
              + ti.matrix[i-1,j+1] - ti.matrix[i+1,j-1]
              - 2*ti.matrix[i+1,j] - ti.matrix[i+1,j+1]);
         l3:=abs(ti.matrix[i,j+1] + 2*ti.matrix[i,j]
              + ti.matrix[i,j-1] - ti.matrix[i+2,j-1]
              - 2*ti.matrix[i+2,j] - ti.matrix[i+2,j+1]);
```

```
        if (l2<l1) or (l2<l3) then  l2:=0;
        if k2>=threshold then k2:=3*k2;
        if l2>=threshold then l2:=3*l2;
        if k2>l2 then
                if k2>peak  then
                            table_out.matrix[i,j]:=peak
                else
                    table_out.matrix[i,j]:=k2
        else
                 if l2>peak  then
                             table_out.matrix[i,j]:=peak
                 else
                     table_out.matrix[i,j]:= l2;
    end;
end;
end;   (* Sobel2a *)
```