1-1-1984

The development and. implementation of edge detection algorithms for shape analysis studies.

Paul F. Hemler

THE DEVELOPMENT AND IMPLEMENTATION OF EDGE DETECTION ALGORITHMS FOR SHAPE ANALYSIS STUDIES

by

Paul F. Hemler

A Thesis

Presented to the Graduate Committee

in Candidacy for the Degree of

Master of Science

in

Electrical Engineering

Lehigh University

1984

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering.

May 4, 1984
Date

Professor in Charge

Chairman of the Department

ACKNOWLEDGEMENTS

Thanks are due to the Electrical an Computer Engineering and Geology Department at Lehigh University for supplying the funds necessary to purchase the equipment used in this project. Many thanks to B. D. Fritchman for his many hours spent debugging the system. Finally thanks is given to R. J. Patch for supplying software support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

An automated facility used to determine the external edge locations of pebbles and sand grains was built. The facility utilizes a microcomputer to control and access a video digitizer. Digitial image processing algorithms that enhance and detect the external edges were implemented in software. The microcomputer stores the detected edges in a disk data file for later shape analysis studies. The facility allows the user to determine the edge locations of up to two dozen pebbles at one time, and with the use of an appropriate lens the system will detect an equal number of sand grain edges. This paper describes the development of this facility with an emphasis on the image processing techniques employed.

I. INTRODUCTION

The goal of this project was to develop an automated image processing facility to do shape analysis studies on pebbles and sand grains for the Geology Department at Lehigh University. The shape of the sand grains can be used to determine the dominant source of sand in erosion problems. Studying samples from different depths reveals climatic and environmental conditions during the period of time that these levels were formed. In addition, the shape information can be used to determine which transport mechanism was most dominent in moving samples to their present locations.

In the past a system which gathered the external edge information manually was employed. The samples were projected onto a digitizer pad, then the outline of the shadow was traced by hand. This method had several very severe limitations when using this data for shape analysis studies. A considerable number of man hours were required to digitize the necessary amount of samples needed to form a meaningful data base. In addition, the edge data was not reproducible and the accuracy of the traced edge was operator dependent.

In the automated system, a camera and a video digitizer

are used to digitize an image containing several different pebbles or sand grain samples. In order to enhance and detect the external edges of the samples, software utilizing different types of digital image processing algorithms was written. This software includes contour tracing, filtering, and thinning algorithms. The software accesses the digital image and determines the X and Y coordinates of the external edge. These edge coordinates are then stored in a disk data file for later shape analysis studies. This data file is the same type of file established when using the manual system, so the shape analysis programs already being used with the manual data can utilize this new data. In addition to automating the method of gathering the edge data, a considerable increase in edge accuracy and reproducibility of the data was achieved.

II. DESIGN CONSIDERATIONS

In order to design an automated image processing facility, several potential problems had to be overcome. An automated method of capturing the video image of several different pebbles or sand grains at one time had to be created and there had to be good contrast between these objects and the background. A frame was designed to be

placed on a table top and was used to position the camera at its focal distance above the objects. The frame allowed the operator to slide the samples directly under the camera. A focal plane of about 12 inches by 12 inches resulted when a standard 16mm television lens was used with the camera. The objects were placed on a white background to ensure good contrast between themselves and the background. The algorithms used to determine the shape of the objects could then be applied to the entire screen allowing about two dozen objects to be processed at one time. In the current facility the objects chosen were pebbles, but the facility could be easily modified to allow the user to study the shapes of sand grains.

The pebbles to be studied were three dimensional objects having certain thickness. When improper lighting was used, shadows around the pebbles were created. No attempt was made to correct for this problem, but two possible solutions follow:

1) Place the pebbles on a transparent plane fixed ~~above~~ a light source consisting of fluorescent lamps.

2) Enclose the frame in white cardboard and place fluorescent lamps around the base of the frame.

It will be shown later that the image processing algorithms did detect these shadows. For accurate shape

analysis information it would be necessary to eliminate these shadows.

SYSTEMS COMPONENTS

The image processing facility consists of a camera, a support frame for the camera, a video display monitor, a computer terminal, a host computer, a mass storage device, and a CAT-800 video digitizer. The video digitizer requires a true 2:1 interlaced analog camera signal. The camera used in this facility was a Panasonic WV-342 black and white closed circuit television camera. Figure 1 shows the relationship between these individual components.

The heart of the system is the CAT-800 video digitizer. The CAT-800 consists of seven standard S-100 computer boards. Included in this system are a flash digitizer, a video display board, and a 256K-byte solid state memory. The flash digitizer will completely digitize an image frame in 1/30 of a second and store the value of the brightness of each pixel in the solid state memory.[1] The contents of this memory is continuously displayed on the black and white television monitor. The brightness values of the pixels can also be stored on a floppy disk for later reference and processing. Once the gray scale

information is stored in memory, an operator can access it, process the data to determine new values, and replace old pixel values with new pixel values.[2] These operations are performed using software which considers the solid state memory to be a 512 horizontal by 480 vertical integer array. The values of the pixels on the display are found by calling subroutines supplied by the manufacture of the digitizer. (See the section on System Software for more details.)

For the CAT-800 to work properly, it requires the host processor to have an IEEE 696-spec. S-100 bus. The processor that was availiable for use in this facility was an IMS 64k-byte CP/M based microcomputer. It was later determined that this computer did not have one of the required signals on the bus, and needed to be modified in order for the CAT-800 to work. The changes to the system included the design of a circuit to generate the phantom memory signal which was not present on the original bus. When this signal is active, the system memory is disabled from the S-100 bus. This signal is necessary because the CAT-800 transfers data through a window in memory located between addresses E000-FFFF. This is an 8k-byte window located at the top of the 64k-byte system address space. This is also the location in memory where the CP/M

operating system resides. If the system memory is not disabled from the bus, and the CAT-800 is writing data, the data will be written over the CP/M operating system causing the system to crash. The phantom memory signal was generated in hardware by nanding the processor write signal (pwr), with the processor data bus in signal (pdbin). Figure 2 shows a circuit layout for this signal. The logic circuit was placed between the the computer mother board, and the computer's system memory. Figure 3 shows the relationship between these components.

If the operator should wish to sequentially access all the pixels on the monitor, timing problems might occur causing the system to crash. A solution that corrects this problem is to move the CP/M operating system out of the CAT-800's address space. This is accomplished by using the MOVCPM program supplied by the manufacturer of the computer system. With MOVCPM.COM on the logged disk, the user types MOVCPM 50. This causes the computer to generate a memory image of a 50k-byte CP/M operating system. The computer does not use the address space above 50k, so if the CAT-800 writes data to this area no problems occur. The drawback from using this method is that a cold boot is required after a program's execution is completed.

DIGITIZER RESOLUTION

The CAT-800 can be operated using either low, mid, or high resolution. The difference between these modes of operation is in the number and size of the pixels displayed on the screen. The solid state memory used for storing the image is partitioned differently for each resolution. The low and mid resolutions are used primarily for generating and displaying color images. Neither of these two modes was used in any of this image analysis work and will not be discussed. The high resolution mode was chosen because it allowed the maximum number of pixels to be addressed. The screen is divided into 512 horizontal by 480 vertical pixels. The pixels are accessed by supplying the X and Y coordinate values of the desired pixels, where:

$$0 \leq X \leq 511 \quad \text{and} \quad 0 \leq Y \leq 479$$

Figure 4 shows the screen layout. Each pixel on the screen is stored as an 8-bit binary number, where each number corresponds to one of 256 different colors, or shades of gray. The range of pixel colors, or the gray scale goes from black (value 0) to white (value 255). Using this range of colors, it is possible to label a pixel by giving it a slightly different gray scale value, while preserving the appearance of the image to an

observer.

SYSTEM SOFTWARE

The CAT-800 video digitizer will operate using any of the following high level languages: Fortran, Pascal, and Basic. Most of the software written by the author was done in Microsoft Fortran for the following reasons:

1) Most people working in the image processing area are familiar with the Fortran Language.
2) Programs can be compiled and executed faster in Microsoft Fortran than using a Basic interpreter.
3) The IMS computer system did not support the Pascal Compilier.

All of the software used by this facility falls into one of two categories. The Utility Software consists of those programs that are considered to be tools and will be discussed first. With the exception of the first two programs discussed in the Utility Software section, all the software described in this paper was developed by the author for use in the enhancement and detection of the external edges of pebbles and sand grains. Following the discussion of the Utility Software is a description of the Image Processing Software containing the image processing algorithms which were implemented.

UTILITY SOFTWARE

The first program below discusses some of the capabilities of the CAT-800 video digitizer. This is not a complete description, but is sufficient to allow the user to perform some basic functions with the digitizer. With the camera turned on and focused in on a set of samples, the gray scale information is stored in the solid state memory by digitizing a frame of the input. The digitization procedure is a subroutine called from a program named DEMO810. This program was supplied by the manufacturer of the digitizer, and is intended to be used in demonstrating some of the capabilities of the digitizer. Figure 5 gives a brief description of the program and lists the commands that can be issued. After executing DEMO810, the first command that should be issued in order to start the digitization procedure is the I-command to initialize the CAT-800. This subroutine prepares the digitizer for subsequent operations, and partitions the memory for the high resolution mode of operation. If it is desired to digitize an image the P-command is issued. The monitor then will display a digitial version of what the camera is focused on. This digitial image should be saved in a disk file for later

reference and processing. Invoking the R-command will cause the program to enter the disk save routine. The operator is asked if it is desired to save a disk file, S-command, or load a disk file, L-command. If the S-command is issued the operator is prompted for a file name. No extension should be given because this routine will automatically append the extension .CAT to all files. After a file name is given, the program will read the contents of the solid state memory and store it in the disk data file. The memory is 256k-bytes long, therefore, the disk must have at least this much unused space on it. If an image is in a disk file and it is desired that it be displayed, the L-command is given. The operator is again prompted for the file name. Figure 6 shows the execution of the steps described above. OPEB is the file name given to the pixel data of the author's original pebble samples. All subsequent operations will be done using the data in this file.

In order for the user to access and control the digitizer from other programs, the manufacturer of the digitizer supplied a program called DGS800.MAC. This program was written in Assembly Language and must be modified and assembled before it can be used. The modifications are required to configure the program for the specific

language desired and the computer system being used. See Appendix A for a complete description of the modifications necessary for this computer system, and the Fortran language.

After the program is assembled, it can be linked to any fortran program that requires the user to access and control the digitizer. A command is issued to the digitizer the same way any subroutine is called. Table 1 lists all of the possible commands. The name of this subroutine file is CATLIB.REL to remind the user that these are the library routines for the CAT-800.

The rest of the programs presented in this section were developed by the author for use as tools in the enhancement and detection of the external edges of pebbles and sand grains. For the purpose of this work it was decided that all of the image processing algorithms would be applied to one pebble. The algorithms can later be adapted to search the screen for different pebbles, then process each pebble seperately to determine the edge locations.

A subset of the data in OPEB was chosen that contained one complete pebble and one partial pebble that had some

"interesting features". An 82 horizontal by 102 vertical pixel rectangle was removed from the original image and stored in a data file using the program STOPIX. STOPIX prompts the user for the minimum X and Y coordinates of the rectangle, then sequentially reads the gray scale values of the pixels and stores these values in a disk data file. The user must supply the name of the data file which can be any CP/M compatible name. The extension .DAT is automatically appended to the file name to remind the user this is a data file. The X and Y locations of the original test sample were:

XMIN = 29 AND YMIN = 169

The pixel data used by the author is stored in a data file named SAMPLE.DAT. The file created by STOPIX is a 42k-byte long data file, so the user must ensure that there is adaquate space on the disk prior to executing this program.

The compliment of STOPIX is a program called REAPIX. This program prompts the user for the data file name, then reads the file and displays the gray scale values on the monitor. CREAPIX is the same as REAPIX, except this program clears the monitor by setting it to a gray scale value of 100 before reading and displaying the data file. Both programs display the data file at the center of the

monitor regardless of where the original data was found. All of the utility and image processing software described above and in the next section process only the image in this center rectangle.

After a program has processed the image at the center of the screen, it may be desirable to further process this new data. In order to make some comparisions between the original, intermediate, and final data, it is possible to simultaneously display all of these images on the monitor. To accommodate this, the screen is divided into twenty-one different locations where an image rectangle can be stored. GRID is a program that draws the boundries of these locations. Figure 7 shows the grid coordinate values. The storage locations are given a reference number ranging from one to twenty-one where twenty-one is the center location. MOVPIX is the name of a program that copies the contents of one location into another location. The user is prompted for the "move from" location, and the "move to" location. The two locations must be an integer in the range of one to twenty-one.

To examine some of the finer details of an image, the image can be enlarged and displayed on the monitor. SCANPLT is the name of a program that stores the gray

scale information of a square in a buffer. The operator is prompted for the X and Y coordinate values of the center of a square and a square size. The new pixel size is determined from the size of the square that is to be enlarged. A large pixel size will be used to enlarge a small square, and a small pixel size will be used to enlarge a large square. The gray scale values stored in the buffer determine the color of the new sized pixel. The enlarged image is displayed on top of the old data on the monitor, so any important displays should be saved prior to executing SCANPLT. A similiar program to SCANPLT is a program called SCAN. This program prints out the locations of a square along with the coorresponding gray scale values. Scan does not display an enlarged version of the square, but prints the actual gray scale values of the specified square. This is useful when it is desired to know the exact gray scale values of the pixels in a small neighborhood around a central pixel.

During a contour tracing the X and Y coordinate values of the contour pixels will be stored in a data file. This data file can be displayed on the monitor by using a program called EDGEPLT. The operator is prompted for the file name that the edge data was stored in, then the center image rectangle is cleared by setting it white

(value 250). The edge data of the original pebble is displayed at the same location it was detected by setting those pixel locations black (value 5). This provides very good contrast between the background and the detected edge, and is useful for examining the high frequency noise and the horizontal scan lines in the image. It may be desired to view the edge data with the original sample to see how close the detected edge is to the actual edge. EDGOVL is the same as EDGEPLT except the central image location is not cleared prior to displaying the detected edges. The operator can move a copy of the original image to the center image location and execute EDGOVL. This program will only set those pixels at the locations given in the data file black.

The final utility program to be discussed is called OVERPRT. This is an overprinting program that is used to obtain a hard copy of the image at the center screen location. The printer generates thirty-two gray scale values by printing characters over other characters. Figure 8 shows the gray scale produced by OVERPRT. This program was writen in Microsoft Basic because the printer is configured to receive control characters from Basic. The software that accesses the digitizer is still configured for Microsoft Fortran, so Basic cannot access

the solid state memory. Since Microsoft Basic can read a datafile formatted by Microsoft Fortran, it is possible to print the information if it is stored in a datafile. DATAFILE is the name of a program that stores the gray scale values of the image at location twenty-one in a data file named HARDOUT. OVERPRT reads the data in the file HARDOUT and produces a hard copy of it. Figure 9 shows a hard copy of the original sample pebble.

IMAGE PROCESSING SOFTWARE

All of the software described in this section processes the image in the center rectangle, location twenty-one. The gray scale information in this location except for a one pixel thickness around the perimeter is defined to be the image plane. The image plane, therefore, is comprised of 80 horizontal by 100 vertical pixels.

A very useful tool used in image processing is the histogram. The histogram of an image gives the number of occurances of each gray scale level present and can be used to determine certain attributes of the image. The class of images considered here makes it possible to obtain a meaningful histogram without using all the pixels in the image plane. PTGET is a program that samples the

brightness of the pixels in the image plane. The program begins in the lower left hand corner of the image, and samples the gray scale values of every fifth row and every fifth column. The sampled pixels are replaced by white pixels so that the operator can see where the samples were obtained. The sampled values are stored in a disk data file that can be called for later reference and processing. The actual histogram of the image is computed by a program called HIST. The operator is prompted for two file names, the first one being the input data file name, that is, the file created by PTGET, and the second is the file where the actual histogram will be stored. HIST reads the data in the first file, sorts it, then opens a file using the second file name and writes the sorted data to it. A listing of the 256 different gray scale values along with the number of occurances of each level is generated. Figure 10 shows this data for the original sample pebble.

When using histogram data it is often more useful to examine a plotted version of the data as opposed to the tabular listing of the actual output data. PLOT is a program that reads the histogram data file and scales the number of occurances of each gray scale value to a range between zero and one. Figure 11 shows a plot of the histogram for the original sample pebble.

The plotted histogram shows that the image is bi-modal, that is, there are a large number of gray scale values concentrated around two peaks. The smaller peak corresponds to the pebble and the larger peak corresponds to the background. The important thing to notice in this histogram is that the entire gray scale is not used. Only 83 out of 256 possible gray scale values are used.

A histogram can be modified so that all or any number of gray scale values will be used. This can be done by mapping the old set of gray scale values to a new set of values. If the modification technique used is linear, all features of the image will remain the same. This type of linear modification will only enhance the image by giving it better contrast. The modification technique used was to linearly map the original 83 gray scale values to 245 new gray scale values. The new gray scale will now be between black (value 5), and white (value 250). Figure 12 shows an illustration of this technique. The mapping of the old gray level to the new level is according to the equation:

NEW COLOR VALUE = (SLOPE)(OLD COLOR VALUE) + INTERCEPT

The slope can be determined as:

$$5 = M(103) + B$$

$$250 = M(186) + B$$

where M = slope and B = intercept

Solving the equations simultaneously gives:

$$M = 2.95 \quad \text{AND} \quad B = -299$$

SO

NEW COLOR VALUE = (2.95)(OLD COLOR VALUE) - 299

From this equation, all the values between 103 and 186 map to values between 5 and 250. This linear histogram modification technique is applied to the image at location twenty-one using the program MODIF. This program requires that three values be supplied at the begining of the program.

MAPTO = 245 This variable represents the number of new gray scale values.

MAPFO = 83 This is the number of old gray scale values.

MAXIN = 186 This is the maximum gray scale value to be mapped.

MODIF calculates the slope and the intercept of the mapping line, then maps each pixel in the image plane to its new value. Figure 13 shows the original sample pebble after MODIF has been applied to it.

The better contrast in the modified image makes some of the features of the original pebble more apparent. Horizontal scan lines have appeared making the dark edge

of the pebble much more jagged than it really is. These lines are noise that the camera introduced while scanning the image. The actual edge of the pebble is not well defined, and appears to be fuzzy or smeared out. This fuzz is a result of the camera not being in focus with the actual edge of the pebble. This is a typical problem that occurs since the pebbles have different thicknesses. It is very difficult to ensure that all of the edges of the pebbles will be in focus during one digitization cycle. In addition, shadows are cast due to the improper illumination of the pebbles adding to the fuzz problem. Figure 14 shows the histogram of the modified sample. The new histogram is similiar in shape to the old histogram, the only difference is that the new histogram is spread out over the entire gray scale axis.

CONTOUR TRACING

The data that the geology department required for their shape analysis studies was the location of the external edge of the pebbles and sand grains. This location was given by specifying the X and Y coordinate values of this edge. This is a classic problem in image processing, and a whole class of algorithms is devoted to detecting edges in images.[3] A discussion of several edge detection

algorithms that were researched and implemented in software follows.

A popular algorithm developed by Roberts [4] uses a non-linear two dimensional differencing operator to enhance and sharpen the edges of an image. This operation is described mathematically as:

$$G(j,k)=((F(j,k)-F(j+1,k+1))^2+(F(j,k+1)-F(j+1,k))^2)^{\frac{1}{2}}$$

A computationally simpler form is:

$$G(j,k)=|F(j,k)-F(j+1,k+1)|+|F(j,k+1)-F(j+1,k)|$$

Both forms of the algorithm executed in a short period of time and gave approximately the same output image. The algorithm tended to blur those regions of the image that consisted of approximately the same color. Those regions that did not consist of the same color, namely edge regions, were emphasized the most. This algorithm basically enhanced the edge but did not actually track it. Since for the purpose of this study it was necessary to produce a data file of the actual edge locations not just an enhanced image this algorithm could not be used.

The following two algorithms implemented used a threshold operator to determine if a pixel was part of the external edge or contour. An algorithm that avoids choosing a specfic threshold value is discussed later. The

brightness of each pixel in the image was compared to a chosen threshold value. If the pixel had a brightness less than this threshold value, it was possible that it was part of the contour. If the pixel had a value greater than this threshold value, it was part of the background. When the histogram of the image is bi-modal, a simple way to choose the threshold value is to pick a value half way between the two modes. Using the original histogram a threshold value of:

$$T = 145$$

was picked for this image. Serious shape distortions could result if the threshold value was picked too low or to high.

The threshold operator could then be incorporated into the contour tracing algorithms, or the image could be pre-processed using the threshold operator prior to the application of the contour tracing algorithms. The latter method of applying the threshold operator and the contour tracing algorithms separately was chosen because it was easier to see the results of each operation individually. The threshold operation was applied to an image by executing the program EQUAL. This program prompts the user for the threshold value than compares all the pixels in the image plane to this value. If a pixel had a value less then the threshold, it was set black (value 5). If

the pixel value was greater than or equal to the threshold, it was set white (value 250). Figure 15 shows the original pebble after EQUAL had been executed where T = 145. This is a bi-level image and it provides excellent contrast between the pebble and the background. The horizontal scan noise in this image is more apparent than before. In looking at an image after EQUAL had been applied to it, single pixels and small clusters of pixels can be seen. All of these types of pixels are considered noise.

The next algorithm implemented was proposed by Duda and Hart,[5] and traced the exterior edge in a counter-clockwise fashion. A 33-point mask was used to determine the next contour pixel given the previous edge pixel location. The search mask was defined as:

33	30	27	26	24	21
32	29	25	23	11	20
31	28	22	4	10	19
		X	3	9	18
		1	2	8	17
		5	6	7	16
		12	13	14	15

where the numbers in the mask corresponded to the order the pattern was searched. The mask was rotated 90 degrees

left or right depending on the mask location of the last determined edge pixel. Since the contour tracing was to be done on a bi-level image, the image plane was scanned vertically to find the first black pixel. The algorithm then determined the color of the next pixel in the search mask. If the pixel was white, it was part of the background and the color of the next search pixel in the mask was determined. If the pixel was black, it was labeled as the next contour pixel and its X and Y coordinate values were appended to the disk data file. This new contour pixel became the current search pixel and the search procedure continued until the current search pixel was the first edge pixel. This algorithm worked well in determining the exterior edges of the sample pebbles, however, a large amount of time was spent storing the 33-point neighborhood in a buffer for each pixel in the image plane.

The final edge detection algorithm implemented was proposed by Pavlidis,[6] and required the least amount of time to execute. This algorithm was implemented using software modules and each is discussed below. CONTOUR is a program that scans the image plane for a white pixel followed by a black pixel. When this combination of pixels was found in the image plane, the black pixel was assumed to be the first pixel of the contour. Program

control then transfered to a subroutine called TRACER that did the actual contour tracing. TRACER examined the 8-point neighborhood around the previously determined contour pixel searching for the next contour pixel. The search pattern of the current contour pixel was determined from the neighborhood location of the previous contour pixel. This method of adaptive searching allowed the algorithm to trace arc's in all directions. When TRACER found a contour pixel, it stored the X and Y coordinate locations in X and Y arrays respectivly. The brightness value of all contour pixels was set to value 100 to allow the operator to watch the tracing on the monitor. The contour tracing continued until the current search pixel was the first contour pixel found while scanning the image plane. When the contour had been completely traced, program control transfered back to CONTOUR which then checked to see how many edge pixels were found by TRACER. If the contour consisted of less then 50 pixels, it was assumed to be one of the small clusters of pixels, or single pixels introduced by the threshold operation, and was discarded. Program control then transfered to a subroutine called ODISK for objects consisting of more than 50 contour pixels. On the first pass to ODISK, the program prompted the operator for a file name in which to store the X and Y coordinate values of the contour. This

file was opened, and the X and Y arrays were written into it. Subsequent calls to ODISK caused the data to be appended into the same file, so that the edge data of all the pebbles was contained in one file. Control then passed to a subroutine called SCRPLT. This subroutine generated a rough plot of the detected contour on the computer terminal.

The detected contours could then be inspected using the program called EDGEPLT. As discussed before, this program cleared the image plane, then set the pixel locations in the data file black. Figure 16 shows the detected contour of the image in Figure 15. The detected contour could be compared to the original pebble by making a copy of the pebble in location twenty-one. Invoking EDGOVL displays the detected edge data as in EDGEPLT, except the background is not set white first. Figure 17 shows the detected contour with the original pebble. Apparent in both of these figures is the horizontal scan noise introduced by the camera. During the contour tracing, these scan lines were traced twice. The first trace found the end of the line, and the second trace put the current search pixel back on the exterior edge. This erroneous contour data increased the size of the data file, and caused problems when this data was used for the shape analysis programs. A discussion is presented next that

eliminates these scan lines.

FILTERING IMAGES

Frequency in the two dimensional image plane is defined to be how rapidly the brightness changes from pixel to pixel. If the brightness of the pixels in a neighborhood is relatively constant, then this neighborhood is a low frequency part of the image. In contrast, if the pixels in a neighborhood possess drastically different colors, then this neighborhood is a high frequency part of the image. Given the a priori knowledge that the pebbles are mounted on a white background, the exterior edge of the pebble is that part of the image where there is a rapid change from white to black. This high frequency part of the image includes the horizontal scan lines, and is apparent in bi-level image of Figure 15. Figure 18 shows an illustration of low, mid, and high frequency edges. Evident in this figure is that the high frequency edge of the bi-level image is analogous to a unit step function in the one dimensional time domain. Both functions are composed of components located from zero frequency to infinite frequency. When the high frequency parts of the image are eliminated the result is a suppression of the scan lines.

One method of eliminating the high frequency components of an image is to low pass filter the image. A filter was implemented in software by completing a discrete convolution on the entire image. The form of the filter operation in the spatial domain is:

$$Q(m1,m2) = \{\{F(n1,n2)H(m1-n1+1,m2-n2+1)$$

for $m1 - 1/2 \leq n1 \leq m1 + 1/2$

$m2 - 1/2 \leq n2 \leq m2 + 1/2$

where Q is the filtered array

F is the input image array

H is the impulse response array

1 is the neighborhood size

Three low pass filters were considered and implemented where the impulse response array for each filter is given below.

LPF1

$$H = 1/9 * \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

LPF2

$$H = 1/10 * \begin{matrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{matrix}$$

LPF3

$$H = 1/16 * \begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix}$$

All of the filters above replace the center pixel with a weighted average of the pixels in a three by three neighborhood.

The frequency response of each filter was dictated by the weighting factor of each pixel in the neighborhood. LPF1 weighed each pixel in the neighborhood by an equal amount, and replaced the center pixel with the average value of the pixels in the neighborhood. This filter had the lowest cutoff frequency, therefore, the resultant image possessed the fewest high frequiencies. LPF2 weighed the center pixel twice as much as the neighborhood pixels. This type of filter retained more input information than the previous filter. LPF3 weighed the center pixel four times as much as the indirect neighbors, and twice as much as the direct neighbors. More original data passes through this filter unattenuated than either of the filters discussed above. LPF3 had the highest frequency cutoff of all the filters implemented, and was used in all subsequent filter operations.

The constant which multiplies all of the impulse response matricies is equal to the reciprocal sum of the matrix elements. This constant was included so that an offset color was not added during the filter operation. Figures

19-21 show the output image for each low pass filter applied to the bi-level sample. Each filter, while different, eliminated the high frequency parts of the image while preserving all of the low frequency components. The result of the filter operation was to supress the horizontal scan noise and to smear the external edges. The smearing occured in all directions, thereby preserving the overall shape of the pebble.

When a bi-level image is low pass filtered the result is a a filtered image that consists of pixels having gray scale values between the original two levels. The background will stay the same color, as will the pixels in the interior of the pebble. The smeared edge pixels are those pixels that have a brightness value between the original two levels. EQUAL was applied to the image of figure 20 with a threshold of:

$$T = 249$$

This threshold value was chosen so that all the pixels in the image except the background pixels would be changed to black. Figure 22 shows all the smeared edge pixels set black (value 5). A comparision of this image to the bi-level image shows the elimination of the horizontal scan lines. This new image has the same shape as the original pebble with the only difference being that this image is scaled slightly larger. The scaling was a result of the

smearing caused by the low pass filter operation, followed by setting the smeared pixels black. The contour tracing algorithm was applied to this image and the detected contour is shown in Figure 23. The contour of the partial pebble was not detected this time because the tracing algorithm searched for a white pixel followed by a black pixel. With the elimination of the scan lines, this criterion was not met, consequentially its edge was not detected. Figure 24 shows the newly detected contour viewed with the modified version of the original sample. As discussed above, this new contour was scaled slightly larger.

REDUCTION OF THE SCALED CONTOUR

Thinning algorithms are widely used in pattern recognition because they offer a way to reduce the amount of data in an image. Application of a thinning algorithm to an image yields a skeleton of the image. The skeleton consists of those pixels that are necessary to maintain the connectivity of the image. All of the pixels that are not required for the connectivity are discarded, thus reducing the amount of data in the image. Typically thinning algorithms are applied to images that contain holes or loops. The result is an image that resembles the original image except all lines having a width of more than one

pixel are replaced by a line having a single pixel thickness. If a solid object is thinned, the resultant object is the same as the original, except that it is scaled down by one pixel. This is because the perimeter pixels are not required to maintain the connectivity of the image. The thinning algorithm implemented by the author searched the image plane for multiple pixels then eliminated them. A multiple pixel is defined to be a pixel which meets one or more of the following conditions:

1) It is traversed more then once during a contour tracing.

2) It has no neighbors in the interior of the region.

3) It has at least one direct neighbor which belongs to the contour but which is not one of its C-neighbors.

The C-neighbors of a pixel are defined as those pixels traced immediately before and immediately after the current pixel using the contour tracing algorithm TRACER.

THIN is a program that does the thinning described above by implementing the three conditions as a series of masks. The program makes four passes across the image plane thinning only certain multiple pixels on each pass. The first pass thins those multiple pixels that are black

(value 5) with a white (value 250) neighbor directly on its right. The second pass thins multiple pixels that have a white neighbor above them. The third pass thins multiple pixels with white neighbors on their left. The final pass thins pixels with a white neighbor below them. Thinning the image in this fashion ensures that only contour pixels will be thinned. On each pass across the image, the neighborhood around each pixel in the image plane is examined. If the neighborhood matches one of the mask conditions, the center pixel is a skeletal pixel, and is labeled with the value of 100. If the neighborhood does not match any of the maskes, it is a multiple pixel and is eliminated.

THIN was applied to the bi-level image and the resultant image is showm in Figure 25. A comparision of these two images reveals that the second image is the same as the first except that it is scaled down by one pixel. The detected contour of the thinned image is shown in Figure 26. Figure 27 shows this detected contour along with the modified version of the original sample. The shape of this detected contour is very similar to the shape of the pebble, but appears to be offset from the dark edge. The actual external edge of the pebble is still not known due to the focusing and shadow problems discussed earlier. A comparision of the detected contour to the first bi-level

image shows that this contour is the same as the original detected contour, except that the horizontal scan lines have been removed.

The thinned image of Figure 25 was thinned again and the result is shown in Figure 28. A comparision of these two images shows that once again just the perimeter pixels have been eliminated, scaling the image down by one more pixel. The contour of this image was traced, and is shown in Figure 29. Figure 30 shows this detected contour along with the modified version of the original. Inspection of this image shows a good agreement between this detected contour, and the dark edge of the pebble. By applying the thinning operation to the solid object, the pixels added during the low pass filter operation were eliminated. The second application of the thinning operation resulted in the elimination of some shadow pixels. Thus, using the low pass filter and the thinning operations together resulted in the elimination of the horzontal scan lines, and reduced the effects of other noise pixels. After these two operations were performed on the image, the detected contour was that of the original pebble. Figure 31 shows the detected contour of the original bi-level image along with the contour of the twice thinned image. The overall shape of both contours

are the same, but one is a scaled version of the other.

ELIMINATION OF THE SPECIFIC THRESHOLD VALUE

An alternate method of determining the location of the original external edge utilizes the location of the smeared edge formed by the low pass filter operation. As discussed before, the low pass filter smears the edge in all directions with the actual external edge located in this fuzz. The following algorithm was developed for the bi-level image, then it was applied to the original pebble eliminating the need to determine a threshold value.

After a bi-level image is low pass filtered, the smeared edge consists of those pixels with a brightness value between the original two levels. The histogram of this type of image will show three distinct regions, one for the pebble, one for the background, and one for the fuzzy edge. The smeared edge of Figure 15 consists of pixels with:

$$6 \leq \text{BRIGHTNESS} \leq 249$$

RANGE is the name of a program that examines the brightness of all the pixels in the image plane. If a pixel has a brightness value inside a set range of values, its brightness is preserved. If a pixel has a brightness value outside of this range, it is set white (value 250).

Figure 32 shows the resultant image after executing RANGE with:

$$6 \leq \text{BRIGHTNESS} \leq 249$$

using the data of Figure 21. Then using EQUAL with:

$$T = 249$$

all of the smeared edge pixels were changed black, (value 5). Figure 33 shows the smeared edge pixels changed black. Then the contour tracing algorithm was applied to this image. Figure 34 shows the detected contour. This figure shows that two contours were detected, an exterior contour, and an interior contour. This was expected because the image was not a solid object, and contained both these contours. What was really desired was the contour of an image with a single pixel thickness. As discussed before, this is called the skeleton of the image, and is found by using the program THIN. Figure 35 shows the resultant image after THIN was applied. There are several parts in this image that have a pixel thickness of more than one pixel, therefore this is not the skeleton of the of the image. THIN was again executed on this new image to remove any multiple pixels in the image plane. Figure 36 shows the resultant image after it was thinned twice. This image has no more mulitple pixels, and is the skeleton of the input image. CTHIN is the name of a program that continuously thins an image

until all multiple pixels are deleted. Figure 34 was thinned using CTHIN, and the resultant image was the same as that obtained earlier using THIN two times. In either case, the resultant image was the skeleton of the input image. The contour tracing program was applied to the skeleton image, and the detected contour was the skeleton. Figure 37 shows the detected contour with the modified original image. A close agreement between the detected contour and the actual contour was achieved.

This algorithm could be applied to an image that is not bi-level. The difference is in the range of values that should be preserved. In the bi-level image the range was well defined because the image contained two known brightness values. After the low pass filter was applied to the bi-level image, the smeared edge was comprised of pixels with a brightness value between the original two levels. The original image was not comprised of two levels, but two sets of levels. The two sets of levels were the two modes in the histogram discussed earlier. From the original histogram of the sample, the pebble consisted of pixels with:

$$103 \leq \text{BRIGHTNESS} \leq 128$$

and the background consisted of pixels with:

$$159 \leq \text{BRIGHTNESS} \leq 186$$

The fuzzy edge consisted of pixels with:

$$129 \leq \text{BRIGHTNESS} \leq 158$$

The original sample was then modified using MODIF to enhance the contrast. The sets of values given above linearly map to new sets of values where the fuzzy edge maps to:

$$82 \leq \text{BRIGHTNESS} \leq 167$$

LPF3 was applied to the modified version of the original pebble with the result shown in Figure 38. RANGE was executed on the above image preserving the range of pixels comprising the fuzzy edge. Figure 39 shows this set of fuzzy edge pixels. This set of pixels was changed black, then thinned until the skeleton remained. Figure 40 shows the skeleton of the fuzzy edge. The detected contour with the original pebble is shown in Figure 41. This algorithm has replaced the need to determine a specfic threshold value by finding a range of threshold values.

SYSTEMS ANALYSIS

This system successfully determined the edge location information of many different pebbles that were placed in the image plane. Successive applications of the algorithms to the same image resulted in exactly the same set of detected edges. This test shows that the system met the original reproducibility criterion when detecting edges.

A close agreement between the actual edge of the pebble and the detected edge is apparent when both images are viewed simultaneously. The detected edge was scaled slightly larger than the actual edge and the fuzzy edge in the original sample was changed black (value 5). Choosing samples of approximately the same thickness would reduce the amount of fuzz caused by the camera not being in focus from one sample to the next. In addition, better lighting would reduce the shadows cast by the pebbles, therefore, reducing more of the fuzz.

FUTURE DEVELOPMENTS

The algorithms presented here were adapted to determine the edge locations of all the pebbles on the display monitor. The biggest problem involved in processing the entire display was the amount of time required for the programs to execute. The low pass filter operation took more than one hour to completely filter the display. A decrease in the time it takes to process the entire screen could be achieved by altering the programs so that the first threshold value required, or the range of smeared edge values, is generated from the software, and not by the operator. A further decrease in processing time would

result if all the necessary image processing programs were linked together and executed with one command.

The data file used to store the edge information of all the pebbles on the screen was quite large. Microsoft Fortran stored one byte for each digit in the number. The edge pixel locations required four digits to establish their positions, and Microsoft Fortran added a number delimiter to seperate the different numbers. This created a data file that was five times as large as the actual amount of data contained in the file. The same is not true of the Pascal MT+ Compilier, which if used would reduce the data file space, and disk access time. These changes if implemented would significantly reduce the amount of time necessary to process the image. This would entail the use of another computer system since the IMS system did not support the Pascal Compiler.

BIBLIOGRAPHY

1. Digital Graphics Systems, Operators Manual for CAT-800 video digitizer, vol. 1, Califorina

2. Digital Graphics Systems, Software Manual, vol. 1, Califorina

3. L.S.Davis, "A Survey of Edge Detection Techniques", Computer Graphics and Image Processing, Vol. 4 1975, p. 248-270.

4. W.K.Pratt, Digital Image Processing, Wiley, 1978.

5. R.O.Duda, P.E.Hart, Classification and Scene Analysis Wiley, 1973.

6. T.Pavlids, Algorithms for Graphics and Image Processing, Computer Science Press, 1982.

TABLE 1

Catoff	Turns off the Cat's video output.
Caton	Turns on the Cat's video output.
Color	Sets the output color.
Dirctv	Turns on the direct video mode.
Extsnc	Enables external synchronization.
Frmgrb	Digitizes a frame of the input.
Getpt	Gets the brightness value of the addressed pixel.
Grymap	Initializes the color map selection.
Init	Initializes the Cat-800 to the selected mode of operation.
Intsnc	Enables the Cat's internal video syncronization source.
Lpactv	Returns a true value if the light pen is active.
Lpread	Returns the X and Y coordinates of the last pixel seen by the light pen.
Lpseen	Returns a true value if the light pen has seen a pixel.
Movcur	Establishes a starting location for subsequent operations.
Mpload	Loads a 256 byte color map.
Mpslct	Selects the color map.
Ndirtv	Disables the direct tv mode.
Newmsk	Enables or disables a group of bits for each pixel.
Outary	Outputs an array of vectors.
Outlin	Draws a line from the pervious cursor location to the specified cursor location.
Outpt	Outputs a pixel with the specified color to the specified location.
Outtxt	Displays a string of text.
Ptwndw	Loads a window of data from a specified data buffer.
Rdwndw	Transfers a window of data to a specified data buffer.
Rotfnt	Establishes the orientation for text.
Scrool	Scrools the display.
Setscr	Fills the display with the specified color.
Settrm	Establishes ab ascii character as the current text deliminator.

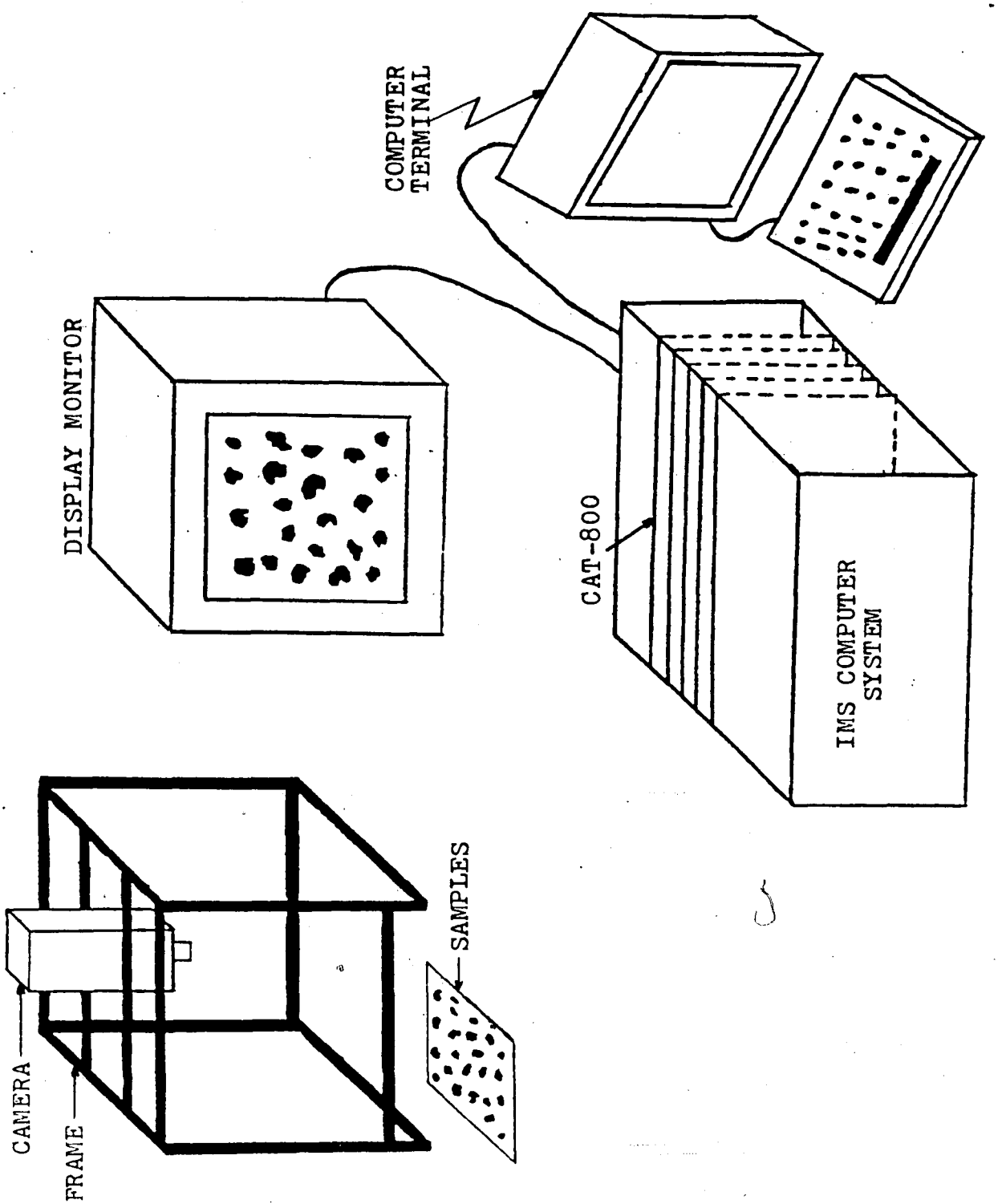

COMPUTER TERMINAL
DISPLAY MONITOR
CAT-800
IMS COMPUTER SYSTEM
SAMPLES
CAMERA
FRAME

FIGURE 1

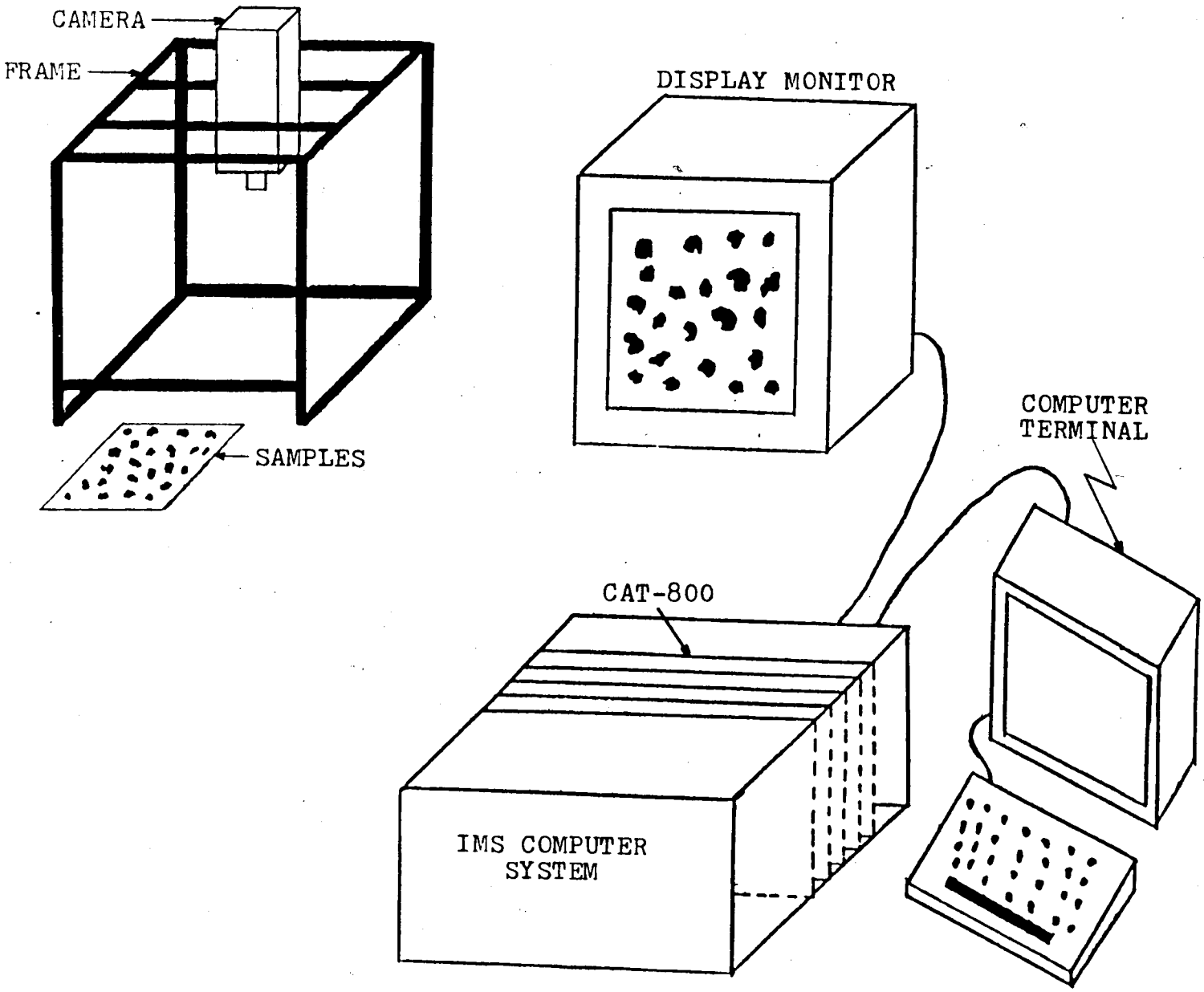

CAMERA
FRAME
SAMPLES
DISPLAY MONITOR
COMPUTER
TERMINAL
CAT-800
IMS COMPUTER
SYSTEM

FIGURE 1

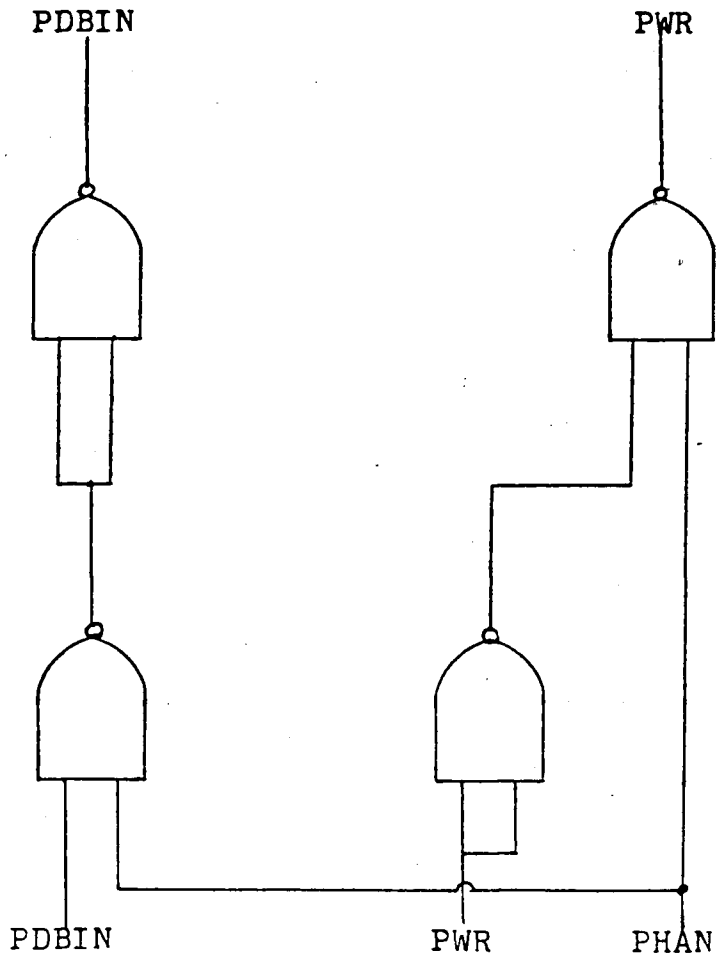


FIGURE 2

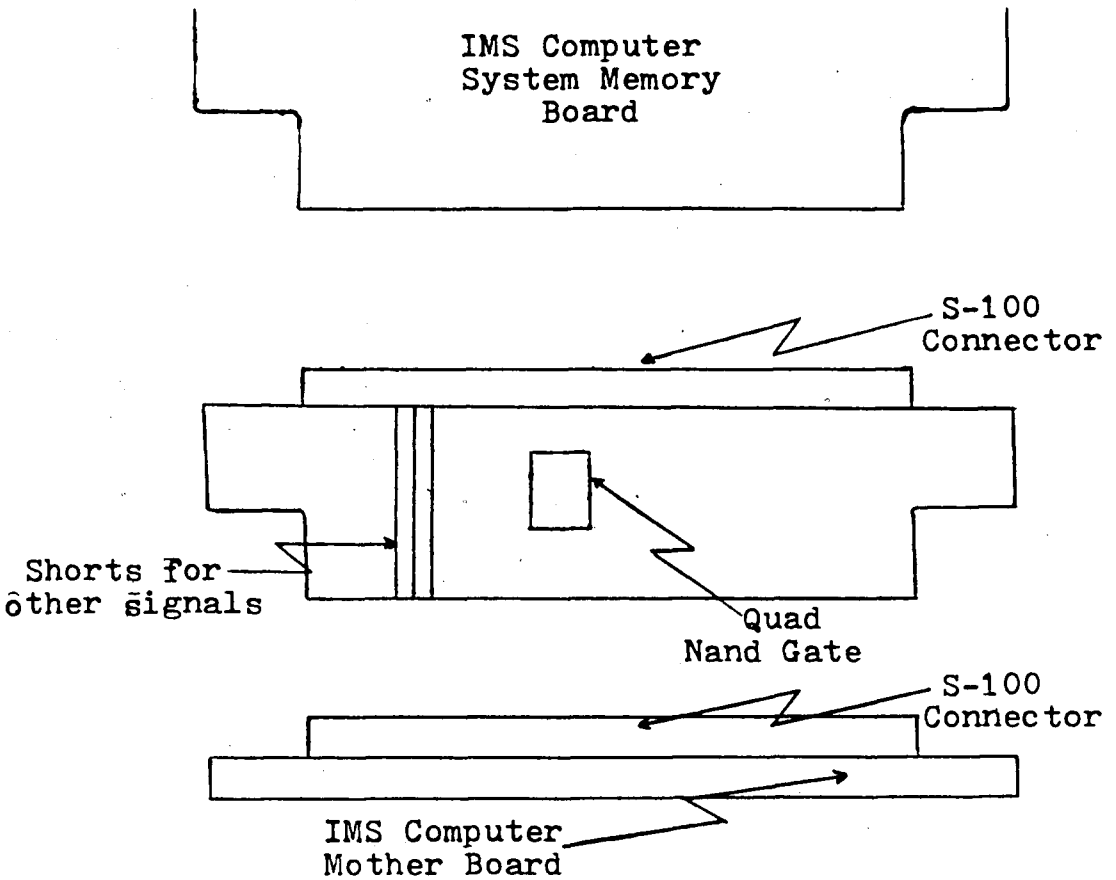

IMS Computer
System Memory
Board
S-100
Connector
Shorts for
other signals
Quad
Nand Gate
S-100
Connector
IMS Computer
Mother Board

FIGURE 3

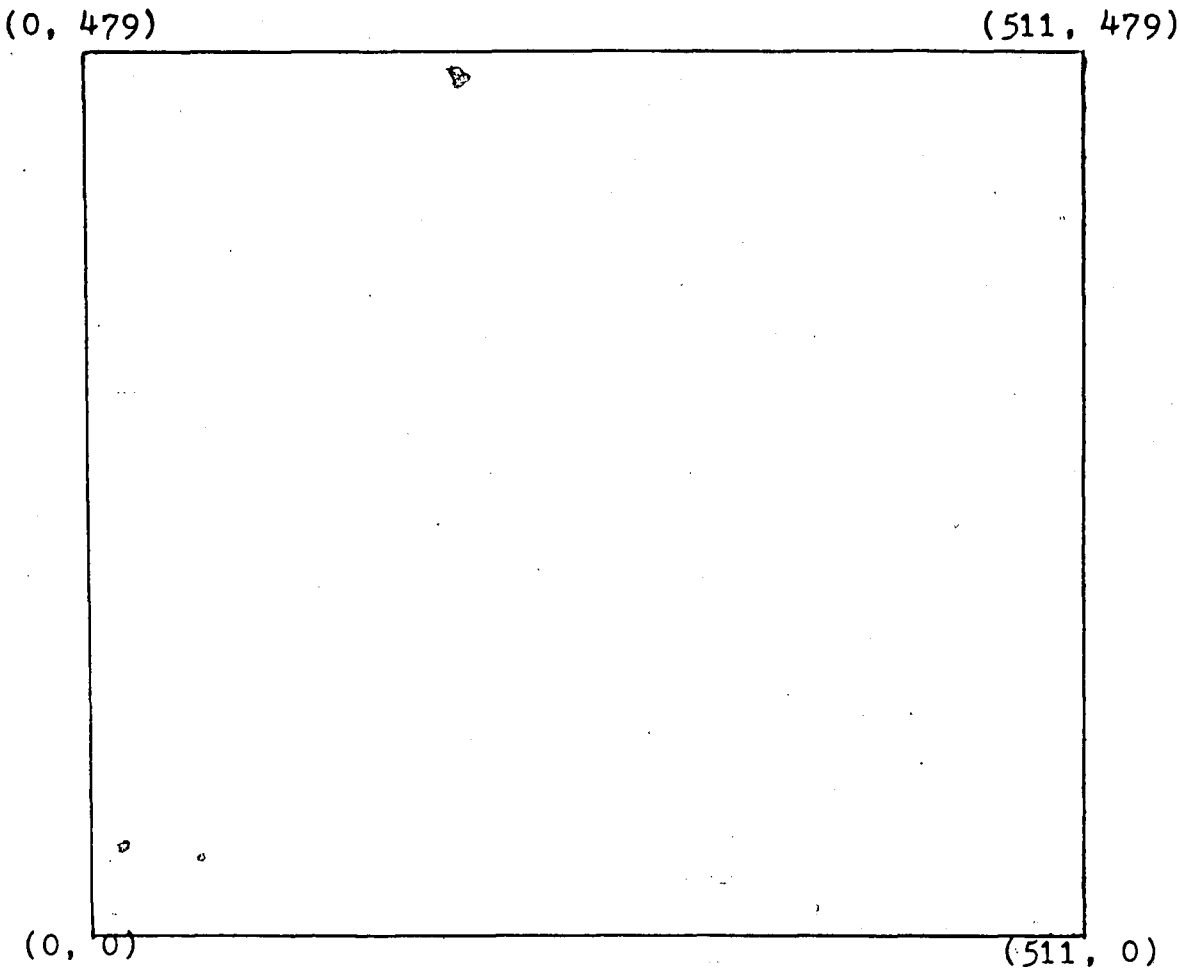


FIGURE 4

```
AVAILABLE COMMANDS FOR THE CAT-800 HIGH RESOLUTION DEMO

  I     Initialize , high res, grey maps.
  B     Blank the digital output.
  E     Enable the digital output.
  V     Toggle the direct video path enable.
  S     Toggle the Synchronization mode.
        (internal/external)
  P     Digitize the video input.
  +     Scroll one line up.
  -     Scroll one line down.
  H     Display this Help message on the console screen.
  C     Set Color for subsequent plotting routines.
  F     Flood the screen with the color number
        last specified ("C").
  D     Drawing routines (More help available).
  T     Text writer.
  A     Color animation routines
  M     Interactive Map composition (more Help available).
  L     Revert to Last map displayed.
  Y     Color filter simulator.
  R     Disk save routines for CAT pictures.
  K     Knitting routines.
  X     Change to Low resolution demo program.
  Z     Zoom.
(TYPE ANY KEY TO RETURN TO MAIN COMMAND LOOP)
```

FIGURE 5

```
DEMO810

        COPYRIGHT 1981 Digital Graphic Systems
              All Rights Reserved

WELCOME TO THE HIGH RES DEMO, TYPE "H" FOR HELP

I,Y,L,H(elp),E,B,+,-,<,>,Z,R,M,K,A,P,S,V,C,D,T,F,G, OR X
? I
INITIALIZE THE CAT TO HIGH RESOLUTION
I,Y,L,H(elp),E,B,+,-,<,>,Z,R,M,K,A,P,S,V,C,D,T,F,G, OR X
? S
EXTERNAL SYNC MODE
I,Y,L,H(elp),E,B,+,-,<,>,Z,R,M,K,A,P,S,V,C,D,T,F,G, OR X
? P
DIGITIZING A FRAME FROM THE VIDEO INPUT
I,Y,L,H(elp),E,B,+,-,<,>,Z,R,M,K,A,P,S,V,C,D,T,F,G, OR X
? R
PICTURE DISK SAVE ROUTINES.
ENTER 'S' FOR SAVE AND 'L' FOR LOAD  - S
PICTURE NAME? OPEB.

I,Y,L,H(elp),E,B,+,-,<,>,Z,R,M,K,A,P,S,V,C,D,T,F,G, OR X
? R
PICTURE DISK SAVE ROUTINES.
ENTER 'S' FOR SAVE AND 'L' FOR LOAD  - L
PICTURE NAME? OPEB

I,Y,L,H(elp),E,B,+,-,<,>,Z,R,M,K,A,P,S,V,C,D,T,F,G, OR X
?
```

FIGURE 6

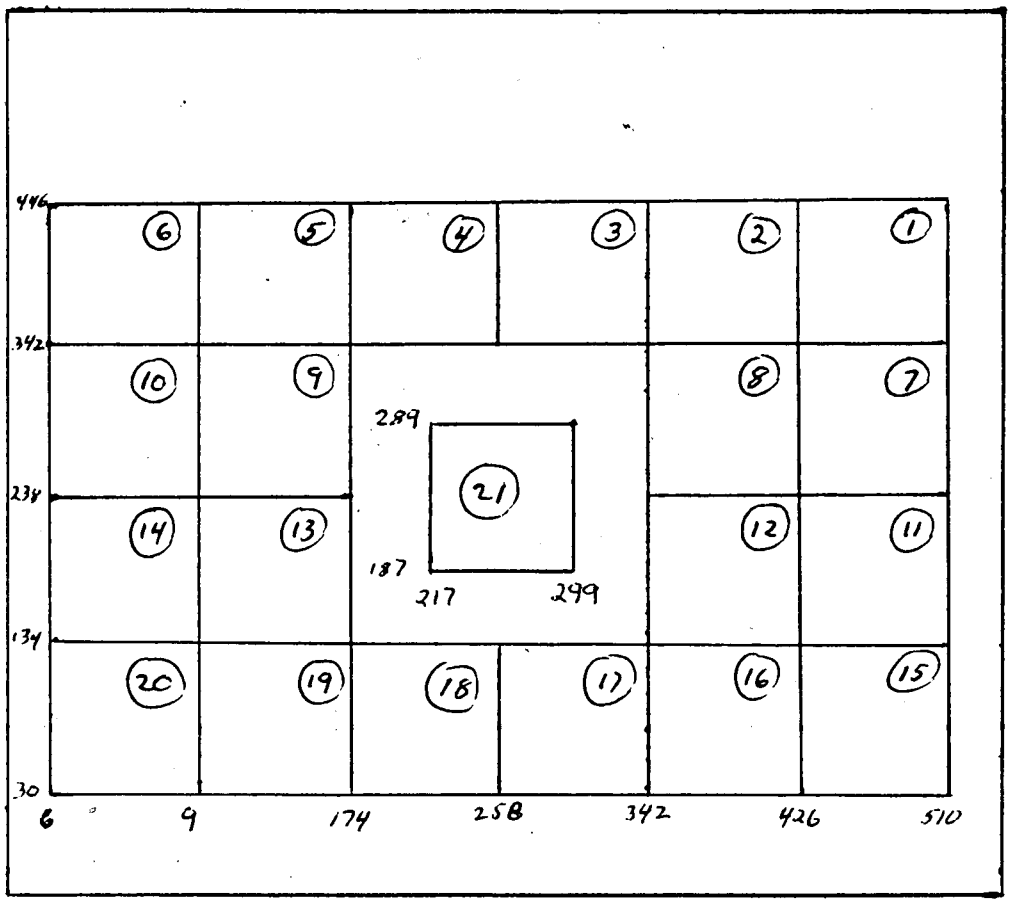

446
342
238
134
30
6
9
174
258
342
426
510
289
187
217
299
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

FIGURE 7

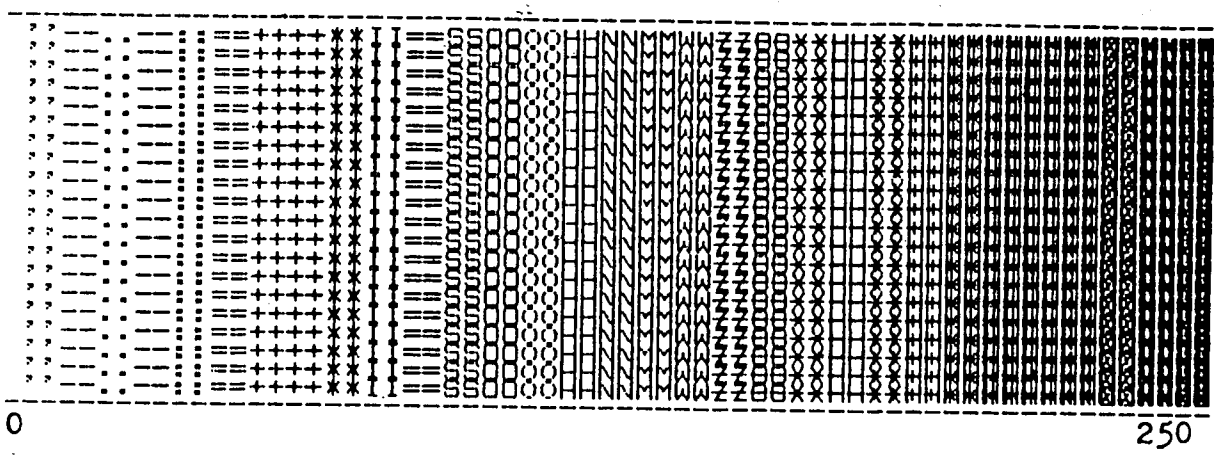


FIGURE 8

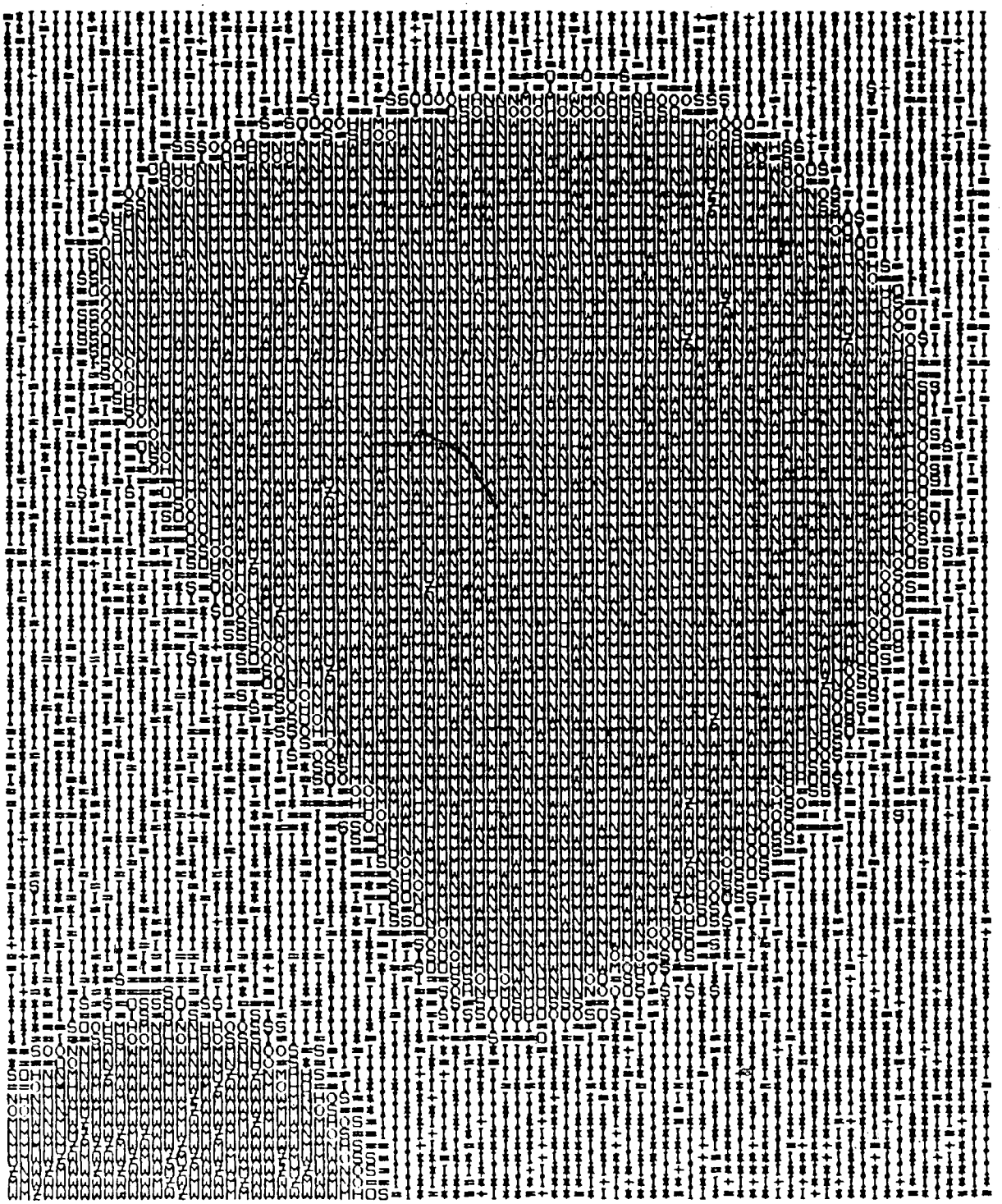


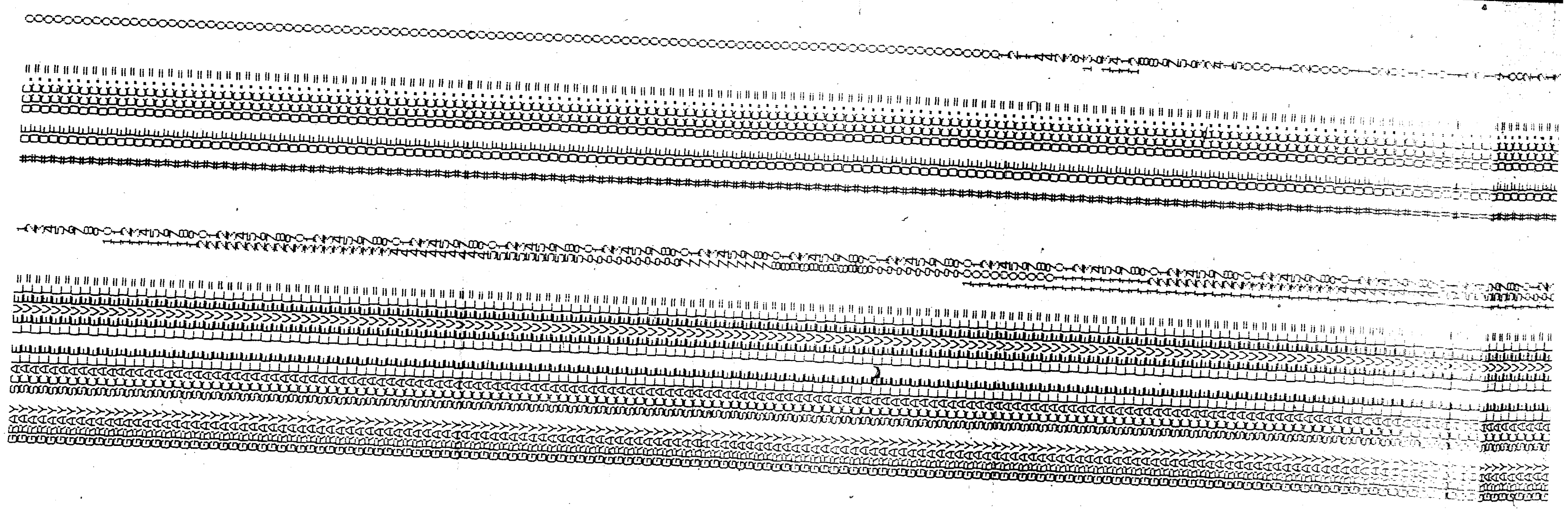
FIGURE 9

GRAY SCALE LEVEL= 1 # OF OCC.= 0
GRAY SCALE LEVEL= 2 # OF OCC.= 0
GRAY SCALE LEVEL= 3 # OF OCC.= 0
GRAY SCALE LEVEL= 4 # OF OCC.= 0
GRAY SCALE LEVEL= 5 # OF OCC.= 0
GRAY SCALE LEVEL= 6 # OF OCC.= 0
GRAY SCALE LEVEL= 7 # OF OCC.= 0
GRAY SCALE LEVEL= 8 # OF OCC.= 0
GRAY SCALE LEVEL= 9 # OF OCC.= 0
GRAY SCALE LEVEL= 10 # OF OCC.= 0
GRAY SCALE LEVEL= 11 # OF OCC.= 0
GRAY SCALE LEVEL= 12 # OF OCC.= 0
GRAY SCALE LEVEL= 13 # OF OCC.= 0
GRAY SCALE LEVEL= 14 # OF OCC.= 0
GRAY SCALE LEVEL= 15 # OF OCC.= 0
GRAY SCALE LEVEL= 16 # OF OCC.= 0
GRAY SCALE LEVEL= 17 # OF OCC.= 0
GRAY SCALE LEVEL= 18 # OF OCC.= 0
GRAY SCALE LEVEL= 19 # OF OCC.= 0
GRAY SCALE LEVEL= 20 # OF OCC.= 0
GRAY SCALE LEVEL= 21 # OF OCC.= 0
GRAY SCALE LEVEL= 22 # OF OCC.= 0
GRAY SCALE LEVEL= 23 # OF OCC.= 0
GRAY SCALE LEVEL= 24 # OF OCC.= 0
GRAY SCALE LEVEL= 25 # OF OCC.= 0
GRAY SCALE LEVEL= 26 # OF OCC.= 0
GRAY SCALE LEVEL= 27 # OF OCC.= 0
GRAY SCALE LEVEL= 28 # OF OCC.= 0
GRAY SCALE LEVEL= 29 # OF OCC.= 0
GRAY SCALE LEVEL= 30 # OF OCC.= 0
GRAY SCALE LEVEL= 31 # OF OCC.= 0
GRAY SCALE LEVEL= 32 # OF OCC.= 0
GRAY SCALE LEVEL= 33 # OF OCC.= 0
GRAY SCALE LEVEL= 34 # OF OCC.= 0
GRAY SCALE LEVEL= 35 # OF OCC.= 0
GRAY SCALE LEVEL= 36 # OF OCC.= 0
GRAY SCALE LEVEL= 37 # OF OCC.= 0
GRAY SCALE LEVEL= 38 # OF OCC.= 0
GRAY SCALE LEVEL= 39 # OF OCC.= 0
GRAY SCALE LEVEL= 40 # OF OCC.= 0
GRAY SCALE LEVEL= 41 # OF OCC.= 0
GRAY SCALE LEVEL= 42 # OF OCC.= 0
GRAY SCALE LEVEL= 43 # OF OCC.= 0
GRAY SCALE LEVEL= 44 # OF OCC.= 0
GRAY SCALE LEVEL= 45 # OF OCC.= 0
GRAY SCALE LEVEL= 46 # OF OCC.= 0
GRAY SCALE LEVEL= 47 # OF OCC.= 0
GRAY SCALE LEVEL= 48 # OF OCC.= 0
GRAY SCALE LEVEL= 49 # OF OCC.= 0
GRAY SCALE LEVEL= 50 # OF OCC.= 0
GRAY SCALE LEVEL= 51 # OF OCC.= 0
GRAY SCALE LEVEL= 52 # OF OCC.= 0
GRAY SCALE LEVEL= 53 # OF OCC.= 0
GRAY SCALE LEVEL= 54 # OF OCC.= 0
GRAY SCALE LEVEL= 55 # OF OCC.= 0
GRAY SCALE LEVEL= 56 # OF OCC.= 0
GRAY SCALE LEVEL= 57 # OF OCC.= 0
GRAY SCALE LEVEL= 58 # OF OCC.= 0
GRAY SCALE LEVEL= 59 # OF OCC.= 0
GRAY SCALE LEVEL= 60 # OF OCC.= 0
GRAY SCALE LEVEL= 61 # OF OCC.= 0
GRAY SCALE LEVEL= 62 # OF OCC.= 0
GRAY SCALE LEVEL= 63 # OF OCC.= 0
GRAY SCALE LEVEL= 64 # OF OCC.= 0
GRAY SCALE LEVEL= 65 # OF OCC.= 0
GRAY SCALE LEVEL= 66 # OF OCC.= 0
GRAY SCALE LEVEL= 67 # OF OCC.= 0
GRAY SCALE LEVEL= 68 # OF OCC.= 0
GRAY SCALE LEVEL= 69 # OF OCC.= 0
GRAY SCALE LEVEL= 70 # OF OCC.= 0
GRAY SCALE LEVEL= 71 # OF OCC.= 0
GRAY SCALE LEVEL= 72 # OF OCC.= 0
GRAY SCALE LEVEL= 73 # OF OCC.= 0
GRAY SCALE LEVEL= 74 # OF OCC.= 0
GRAY SCALE LEVEL= 75 # OF OCC.= 0
GRAY SCALE LEVEL= 76 # OF OCC.= 0
GRAY SCALE LEVEL= 77 # OF OCC.= 0
GRAY SCALE LEVEL= 78 # OF OCC.= 0
GRAY SCALE LEVEL= 79 # OF OCC.= 0
GRAY SCALE LEVEL= 80 # OF OCC.= 0
GRAY SCALE LEVEL= 81 # OF OCC.= 0
GRAY SCALE LEVEL= 82 # OF OCC.= 0
GRAY SCALE LEVEL= 83 # OF OCC.= 0
GRAY SCALE LEVEL= 84 # OF OCC.= 0
GRAY SCALE LEVEL= 85 # OF OCC.= 0
GRAY SCALE LEVEL= 86 # OF OCC.= 0
GRAY SCALE LEVEL= 87 # OF OCC.= 0
GRAY SCALE LEVEL= 88 # OF OCC.= 0
GRAY SCALE LEVEL= 89 # OF OCC.= 0
GRAY SCALE LEVEL= 90 # OF OCC.= 0
GRAY SCALE LEVEL= 91 # OF OCC.= 0
GRAY SCALE LEVEL= 92 # OF OCC.= 0
GRAY SCALE LEVEL= 93 # OF OCC.= 0
GRAY SCALE LEVEL= 94 # OF OCC.= 0
GRAY SCALE LEVEL= 95 # OF OCC.= 0
GRAY SCALE LEVEL= 96 # OF OCC.= 0
GRAY SCALE LEVEL= 97 # OF OCC.= 0
GRAY SCALE LEVEL= 98 # OF OCC.= 0
GRAY SCALE LEVEL= 99 # OF OCC.= 0
GRAY SCALE LEVEL= 100 # OF OCC.= 0
GRAY SCALE LEVEL= 101 # OF OCC.= 0
GRAY SCALE LEVEL= 102 # OF OCC.= 0
GRAY SCALE LEVEL= 103 # OF OCC.= 0
GRAY SCALE LEVEL= 104 # OF OCC.= 1
GRAY SCALE LEVEL= 105 # OF OCC.= 1
GRAY SCALE LEVEL= 106 # OF OCC.= 4
GRAY SCALE LEVEL= 107 # OF OCC.= 4
GRAY SCALE LEVEL= 108 # OF OCC.= 3
GRAY SCALE LEVEL= 109 # OF OCC.= 7
GRAY SCALE LEVEL= 110 # OF OCC.= [illegible]
GRAY SCALE LEVEL= 111 # OF OCC.= [illegible]
GRAY SCALE LEVEL= 112 # OF OCC.= [illegible]
GRAY SCALE LEVEL= 113 # OF OCC.= 9
GRAY SCALE LEVEL= 114 # OF OCC.= [illegible]
GRAY SCALE LEVEL= 115 # OF OCC.= 16
GRAY SCALE LEVEL= 116 # OF OCC.= [illegible]
GRAY SCALE LEVEL= 117 # OF OCC.= 14
GRAY SCALE LEVEL= 118 # OF OCC.= 11
GRAY SCALE LEVEL= 119 # OF OCC.= 8
GRAY SCALE LEVEL= 120 # OF OCC.= 8
GRAY SCALE LEVEL= 121 # OF OCC.= [illegible]
GRAY SCALE LEVEL= 122 # OF OCC.= 2
GRAY SCALE LEVEL= 123 # OF OCC.= 2
GRAY SCALE LEVEL= 124 # OF OCC.= 6
GRAY SCALE LEVEL= 125 # OF OCC.= 7
GRAY SCALE LEVEL= 126 # OF OCC.= 4
GRAY SCALE LEVEL= 127 # OF OCC.= 1
GRAY SCALE LEVEL= 128 # OF OCC.= 0
GRAY SCALE LEVEL= 129 # OF OCC.= 0
GRAY SCALE LEVEL= 130 # OF OCC.= 0
GRAY SCALE LEVEL= 131 # OF OCC.= 0
GRAY SCALE LEVEL= 132 # OF OCC.= 1
GRAY SCALE LEVEL= 133 # OF OCC.= 0
GRAY SCALE LEVEL= 134 # OF OCC.= 0
[illegible]

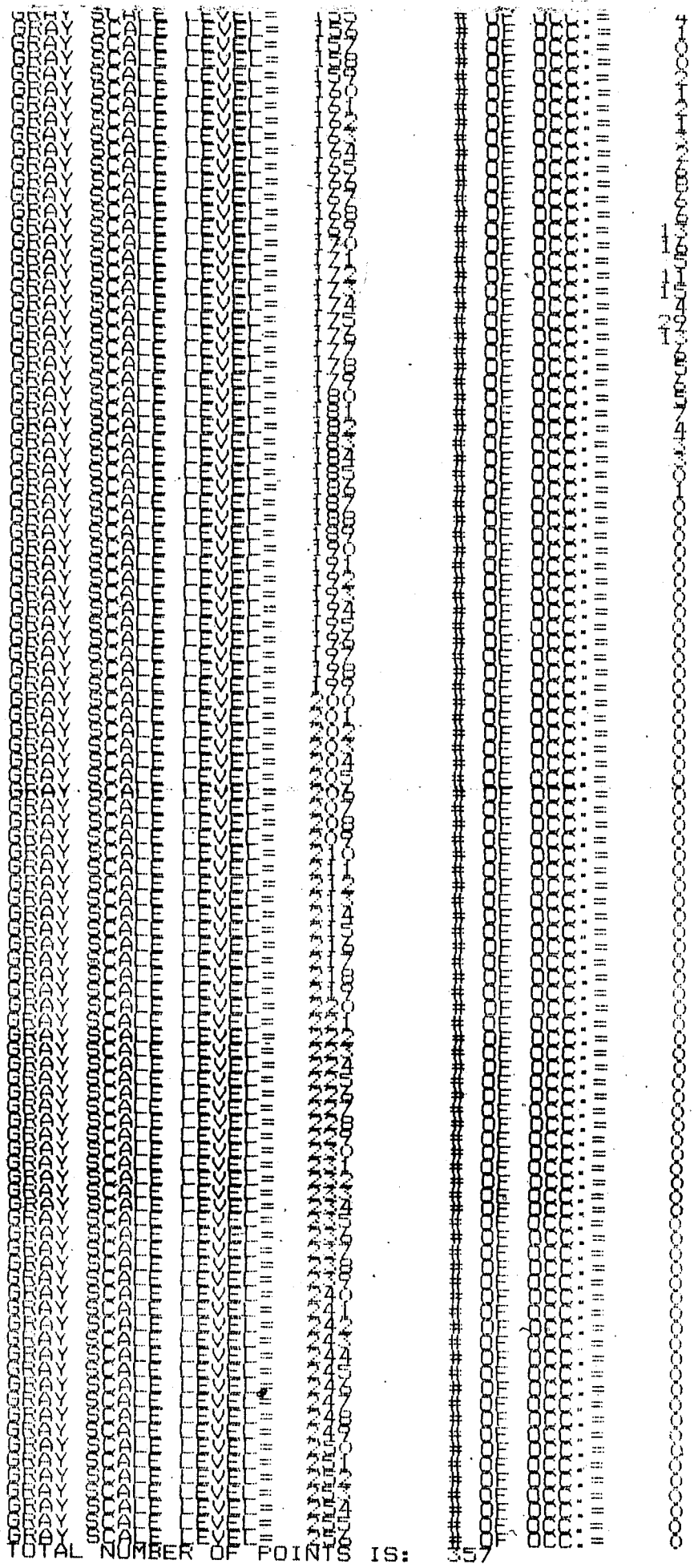
MICRODEX CORRECTION GUIDE (M-9)

CORRECTION

The preceding document has been re-photographed to assure legibility and its image appears immediately here-after.



[redacted]



TOTAL NUMBER OF POINTS IS: 357

DATA FILE HORIG.DAT
0
.5
1

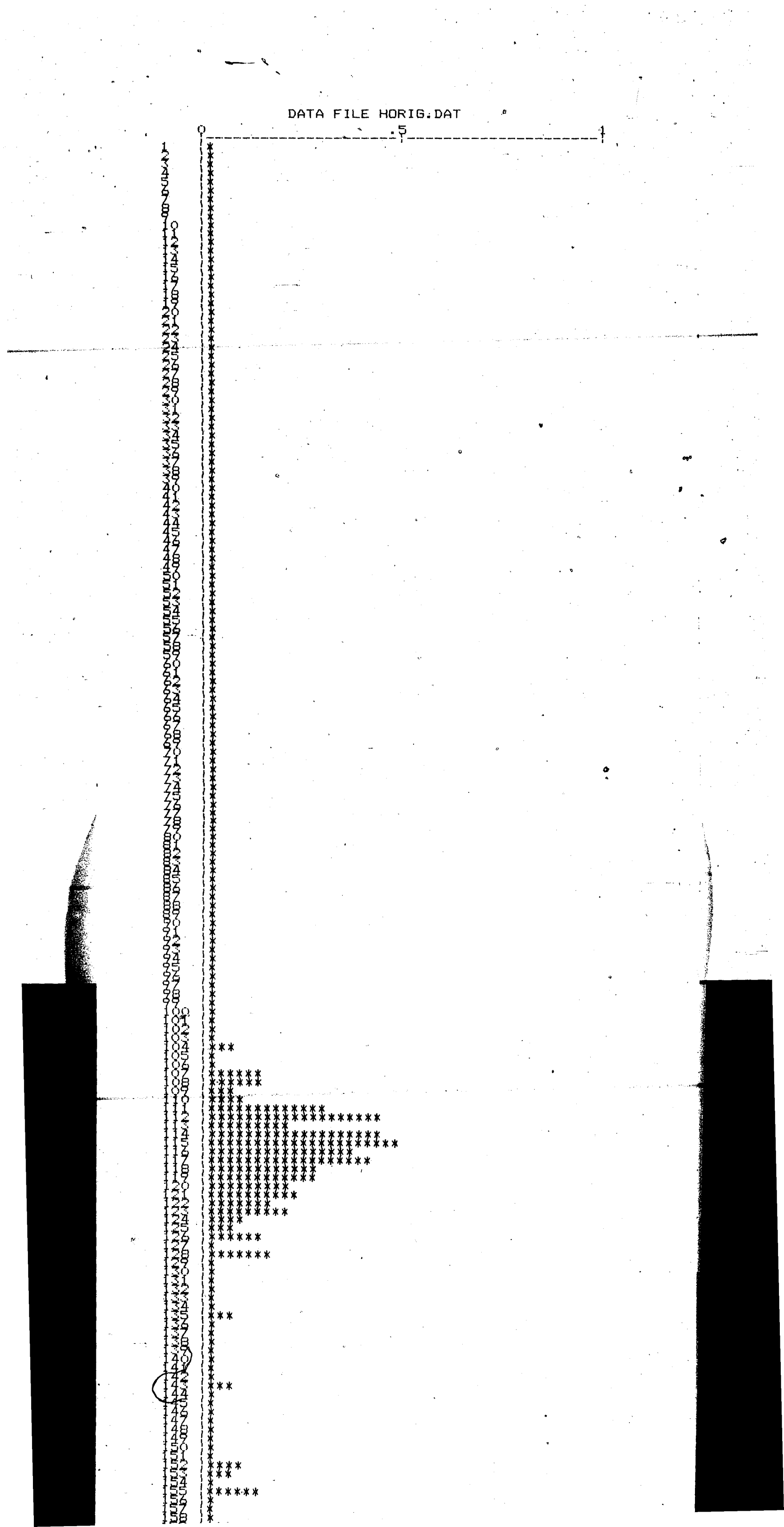

DATA FILE HORIG.DAT
0
.5
1

FIGURE 11

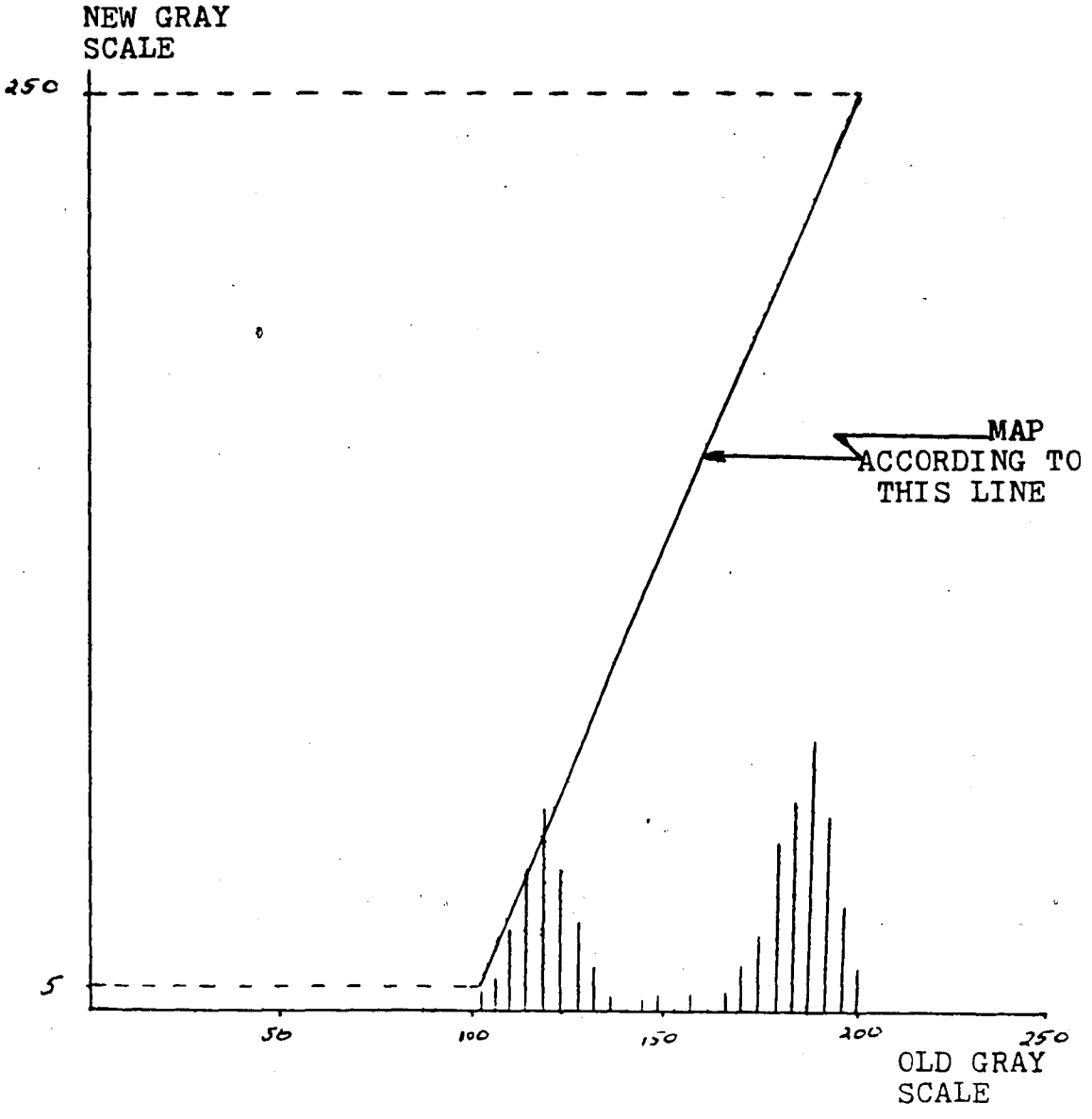

NEW GRAY
SCALE
250
5
50
100
150
200
250
OLD GRAY
SCALE
MAP
ACCORDING TO
THIS LINE

FIGURE 12

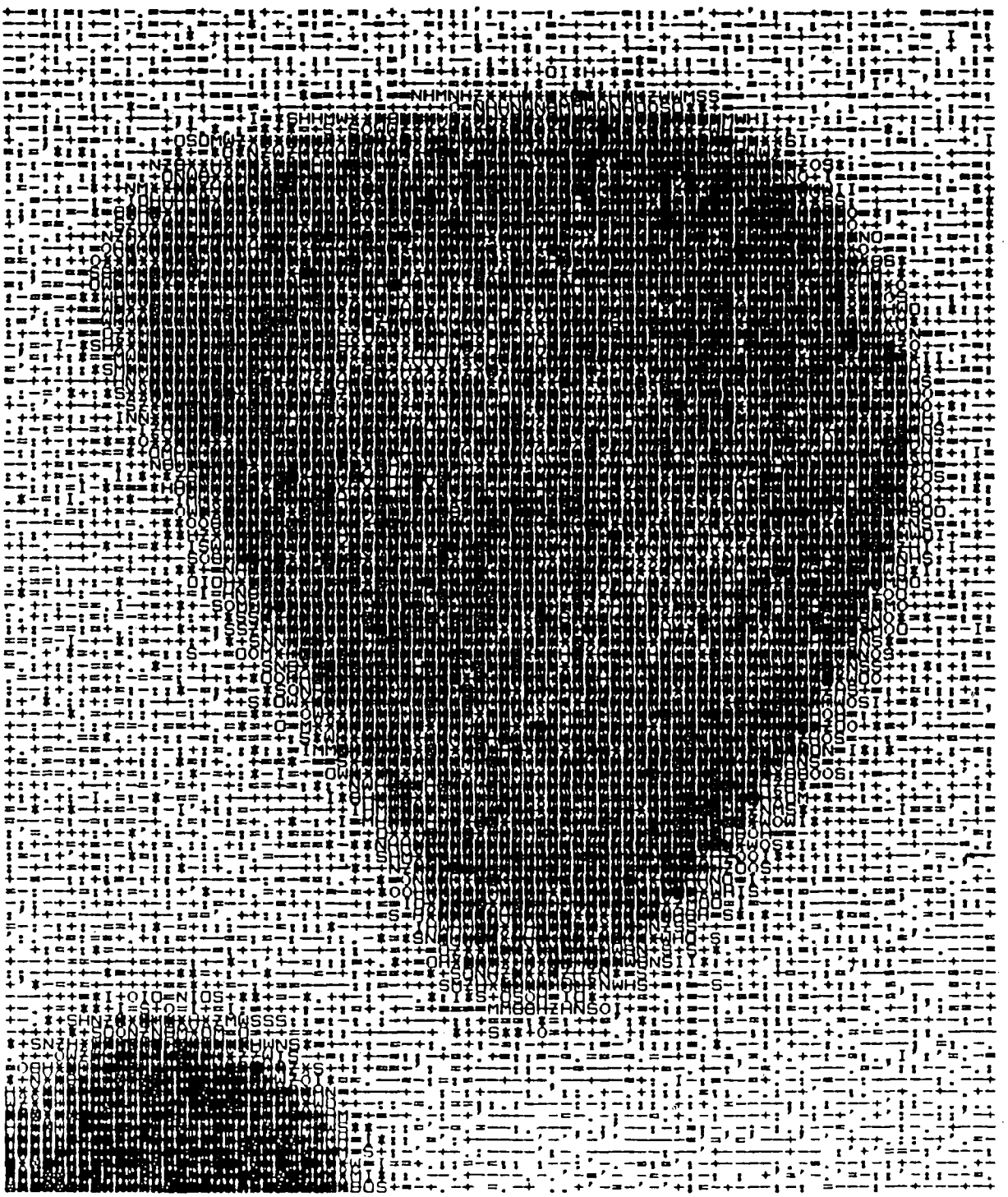
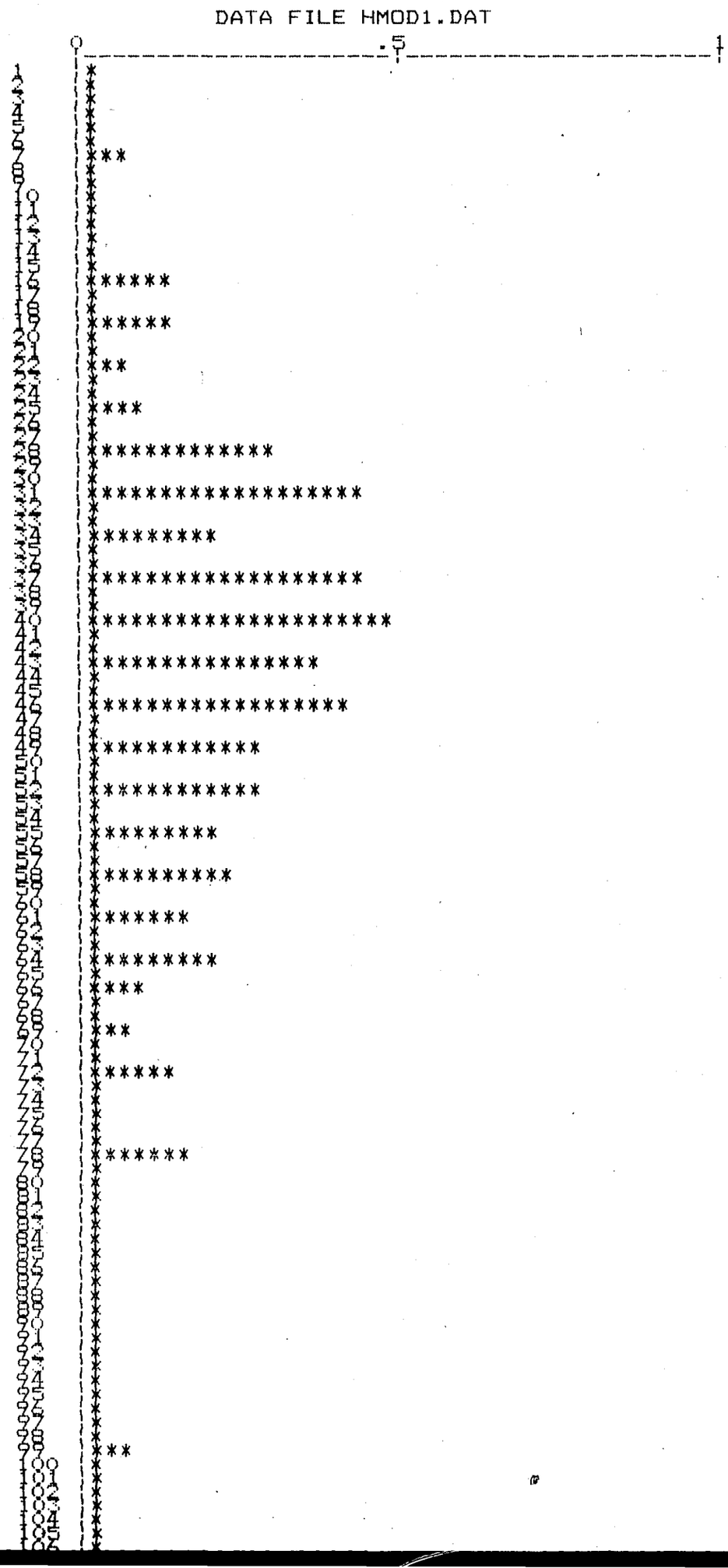
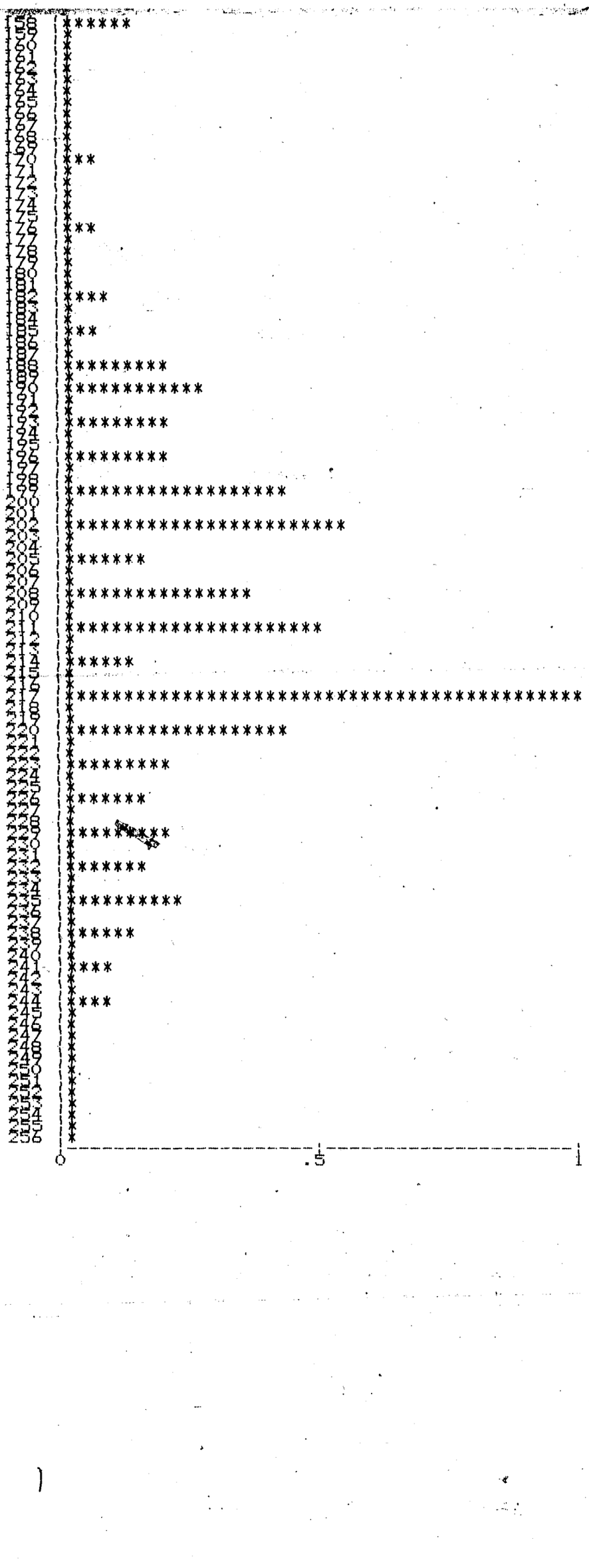


FIGURE 13


DATA FILE HMOD1.DAT
0
.5
1

DATA FILE HMOD1.DAT


0
.5
1

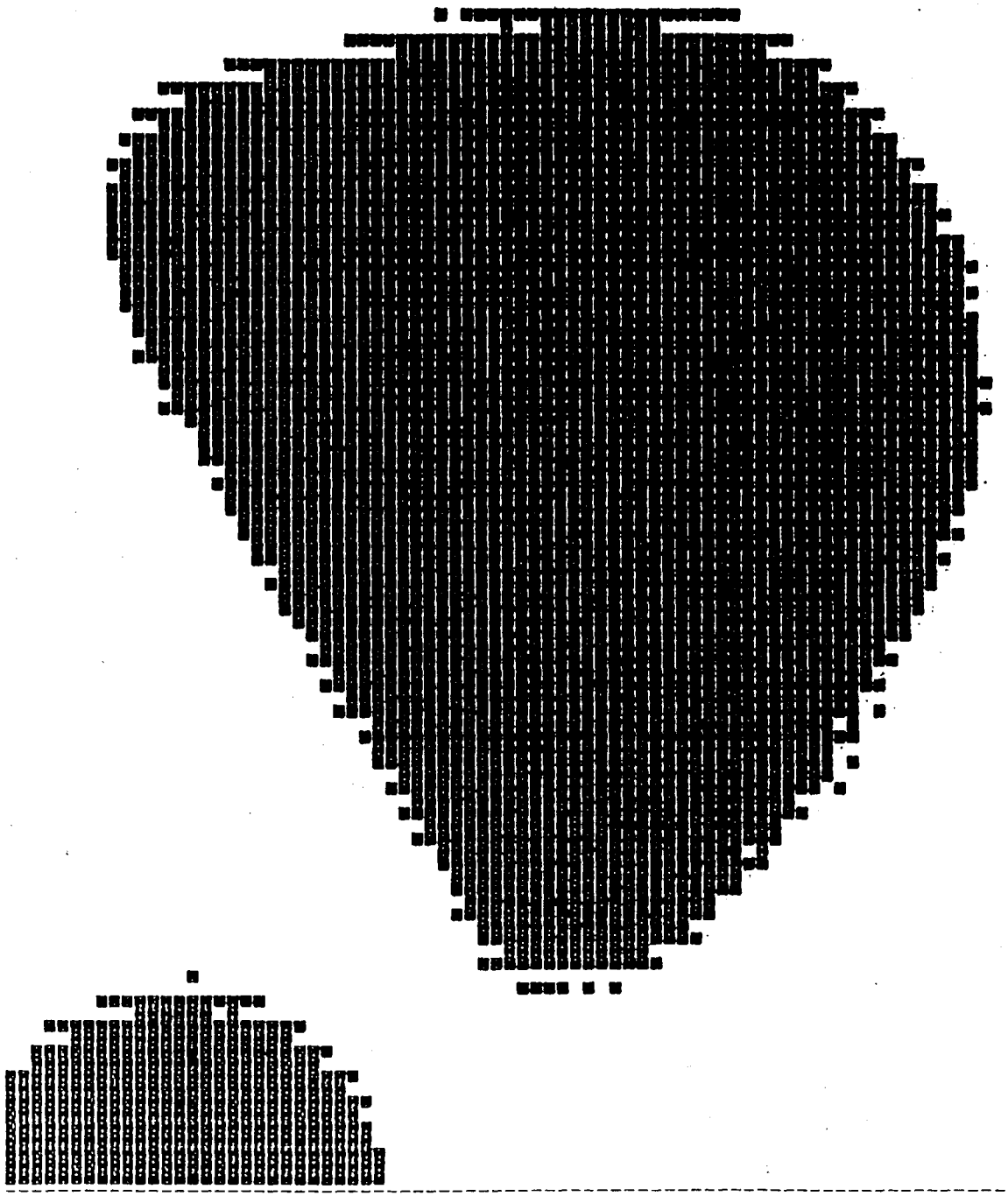


FIGURE 15

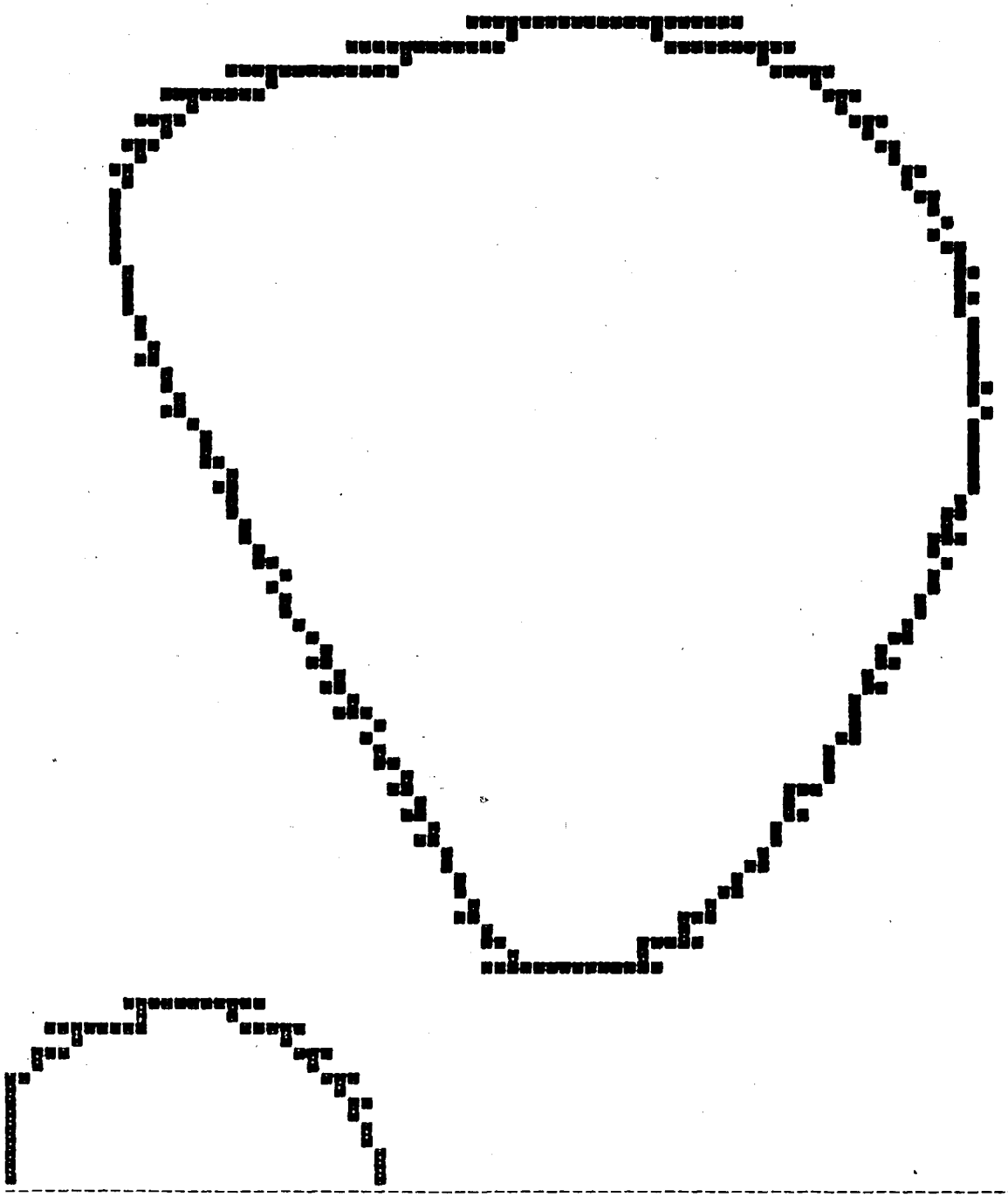


FIGURE 16

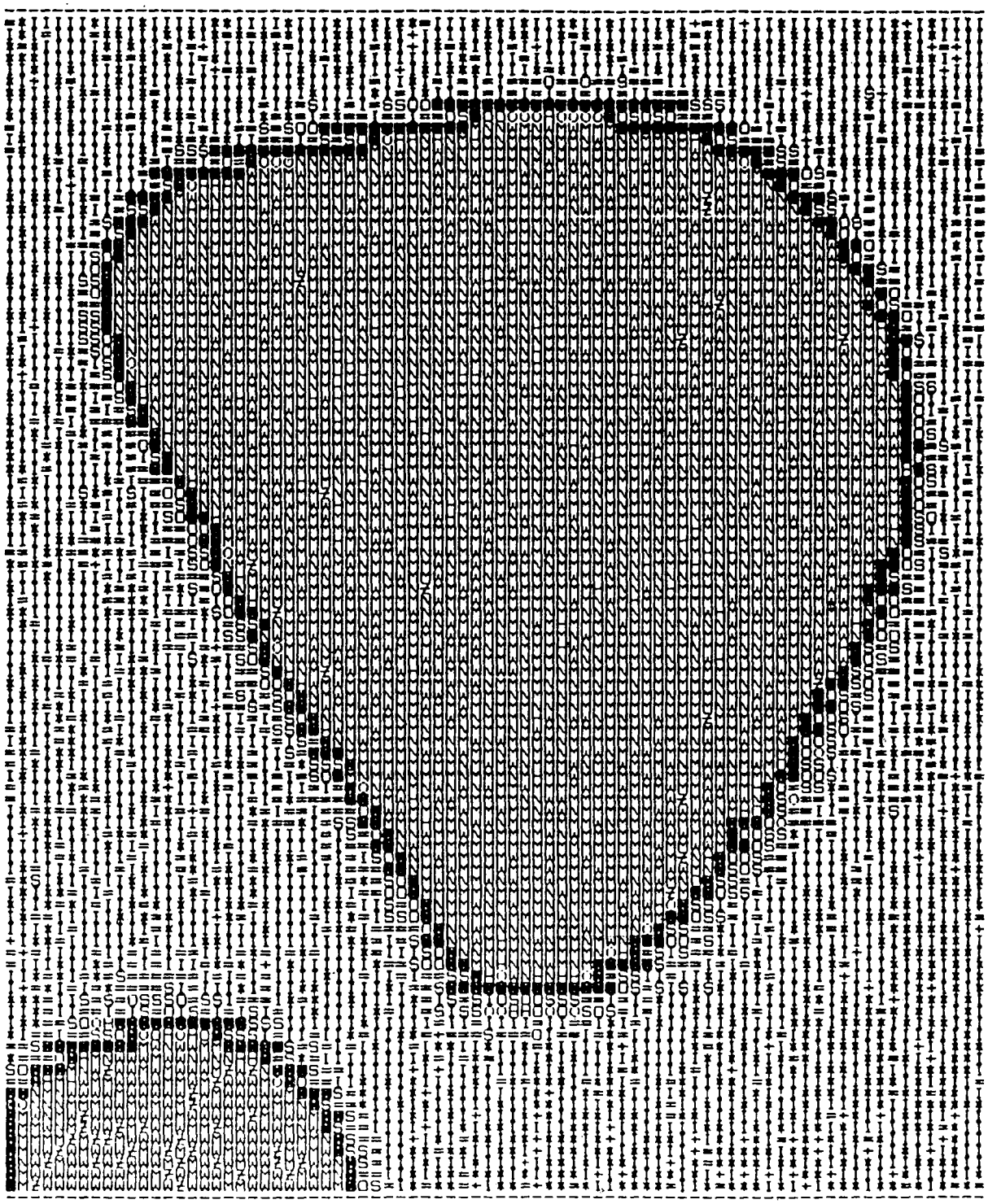
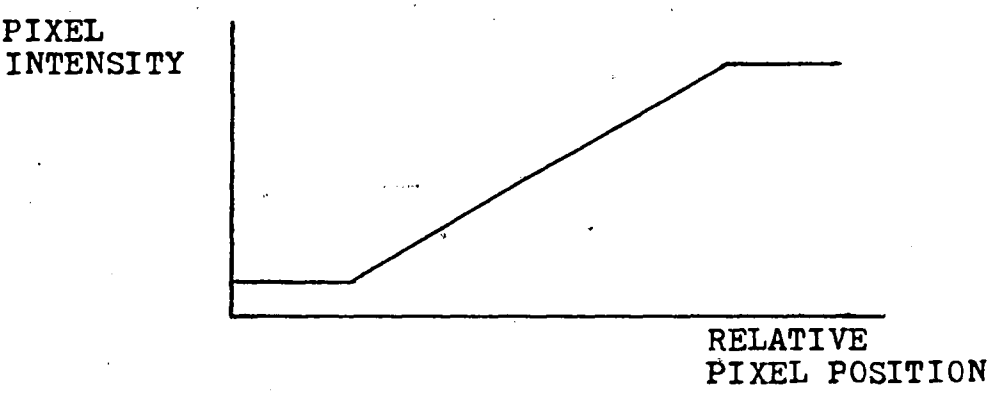
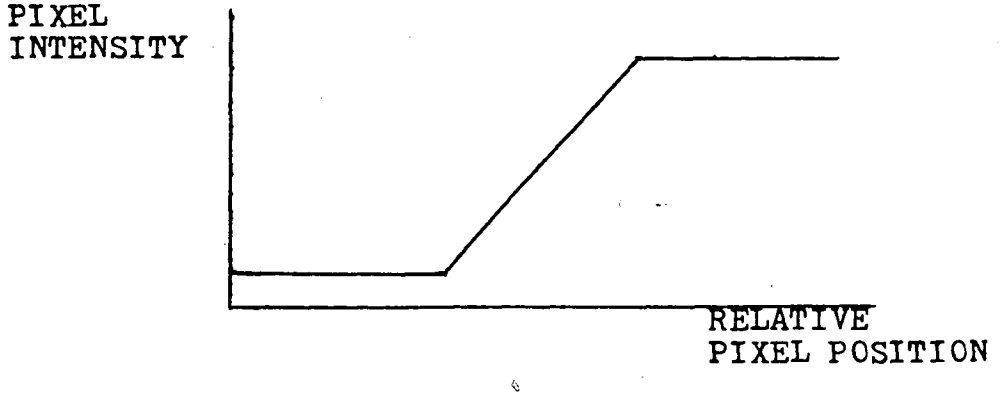


FIGURE 17


PIXEL
INTENSITY
RELATIVE
PIXEL POSITION


PIXEL
INTENSITY
RELATIVE
PIXEL POSITION

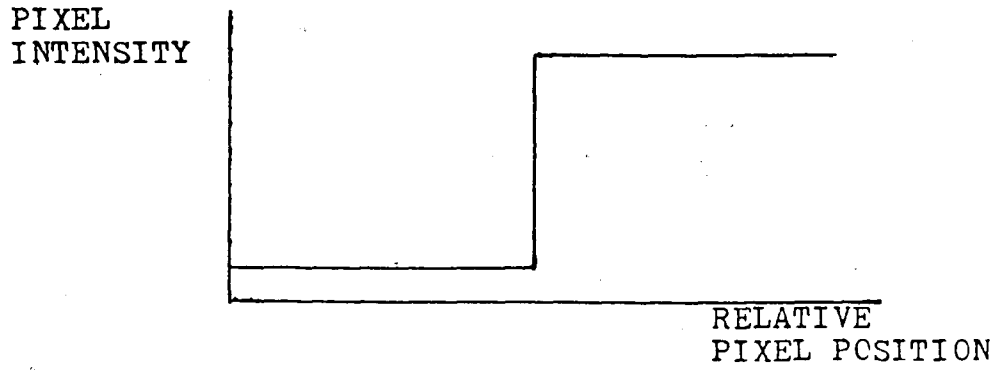

PIXEL
INTENSITY
RELATIVE
PIXEL POSITION

FIGURE 18

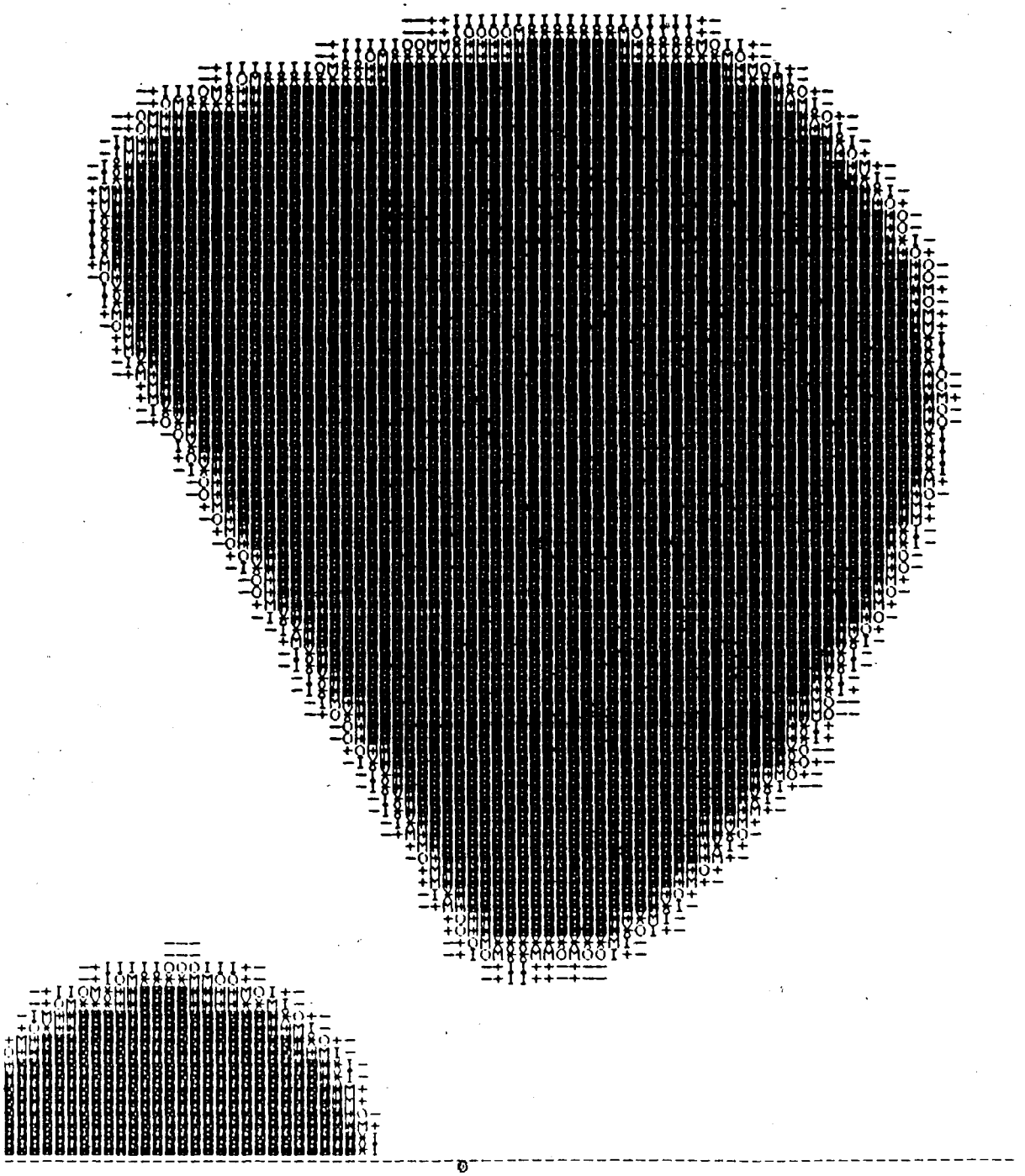


FIGURE 19

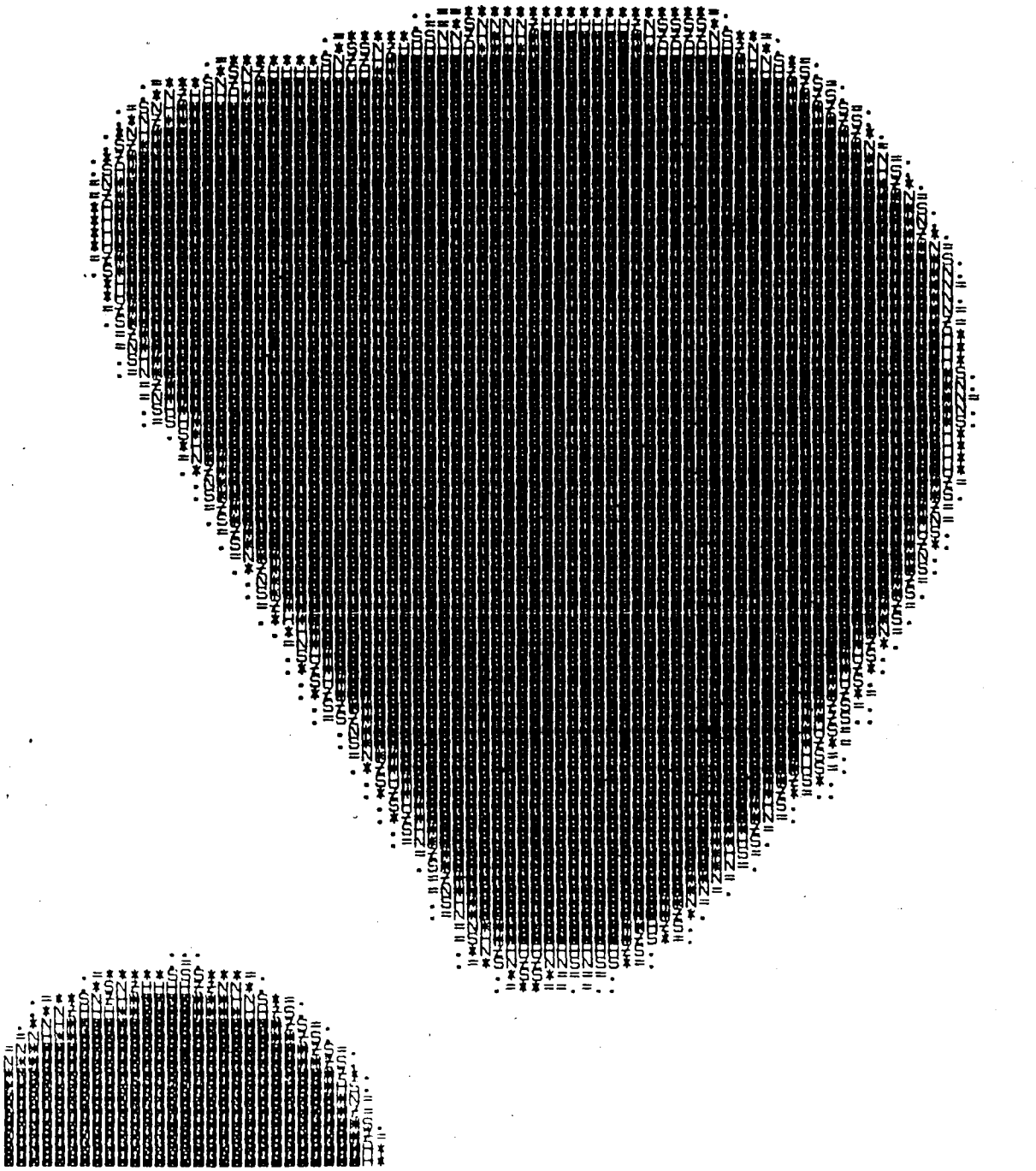


FIGURE 20

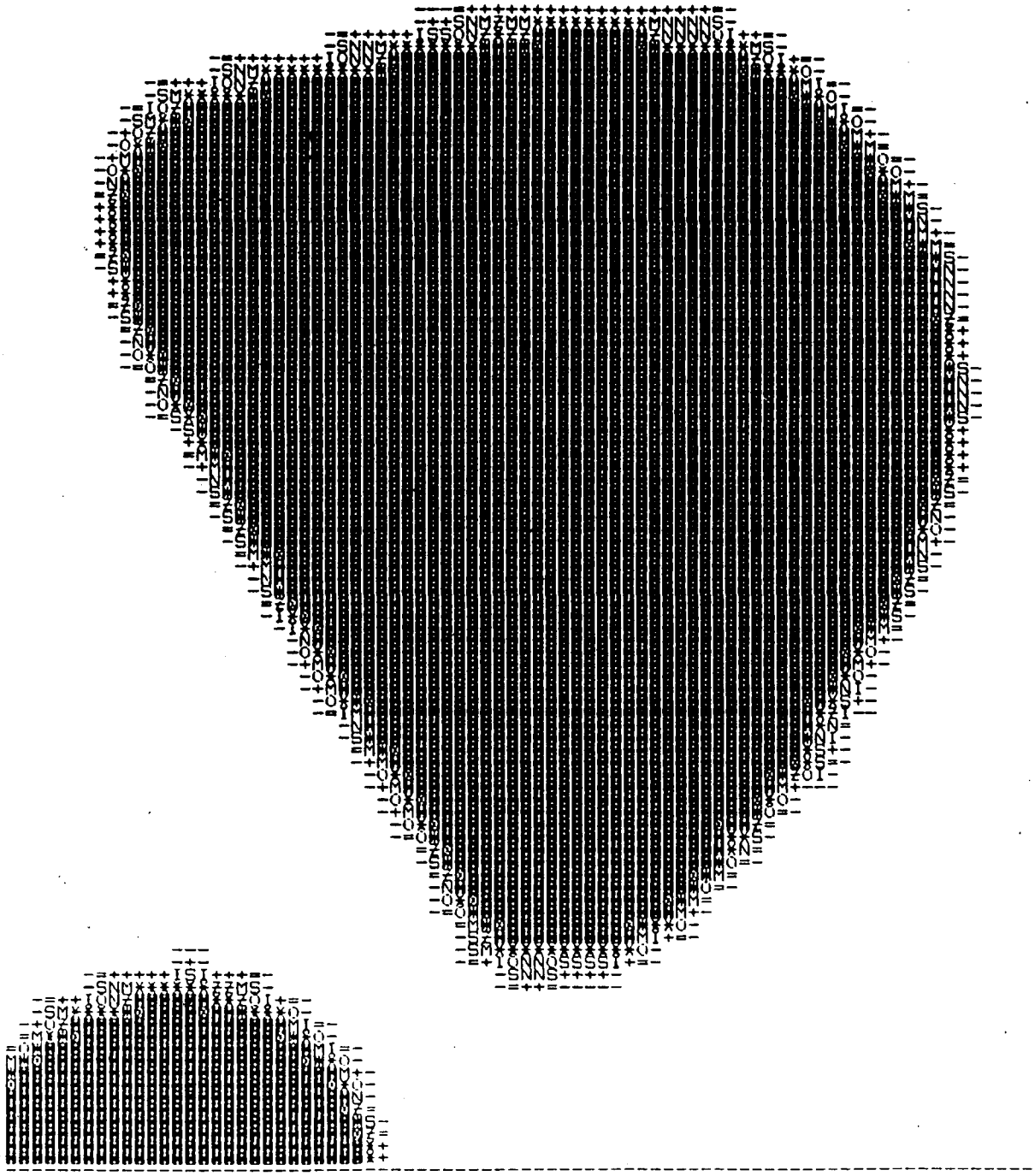


FIGURE 21

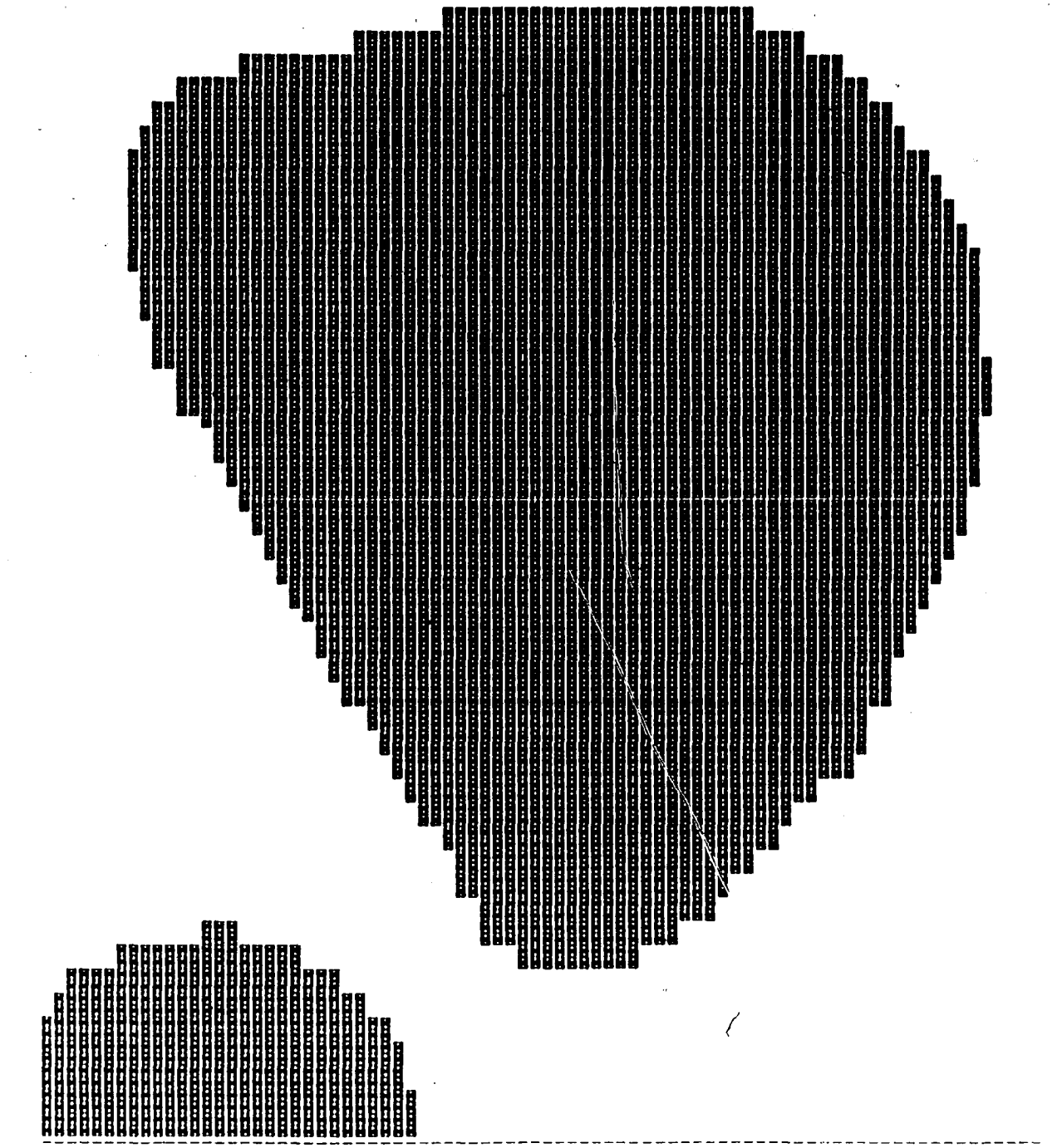


FIGURE 22

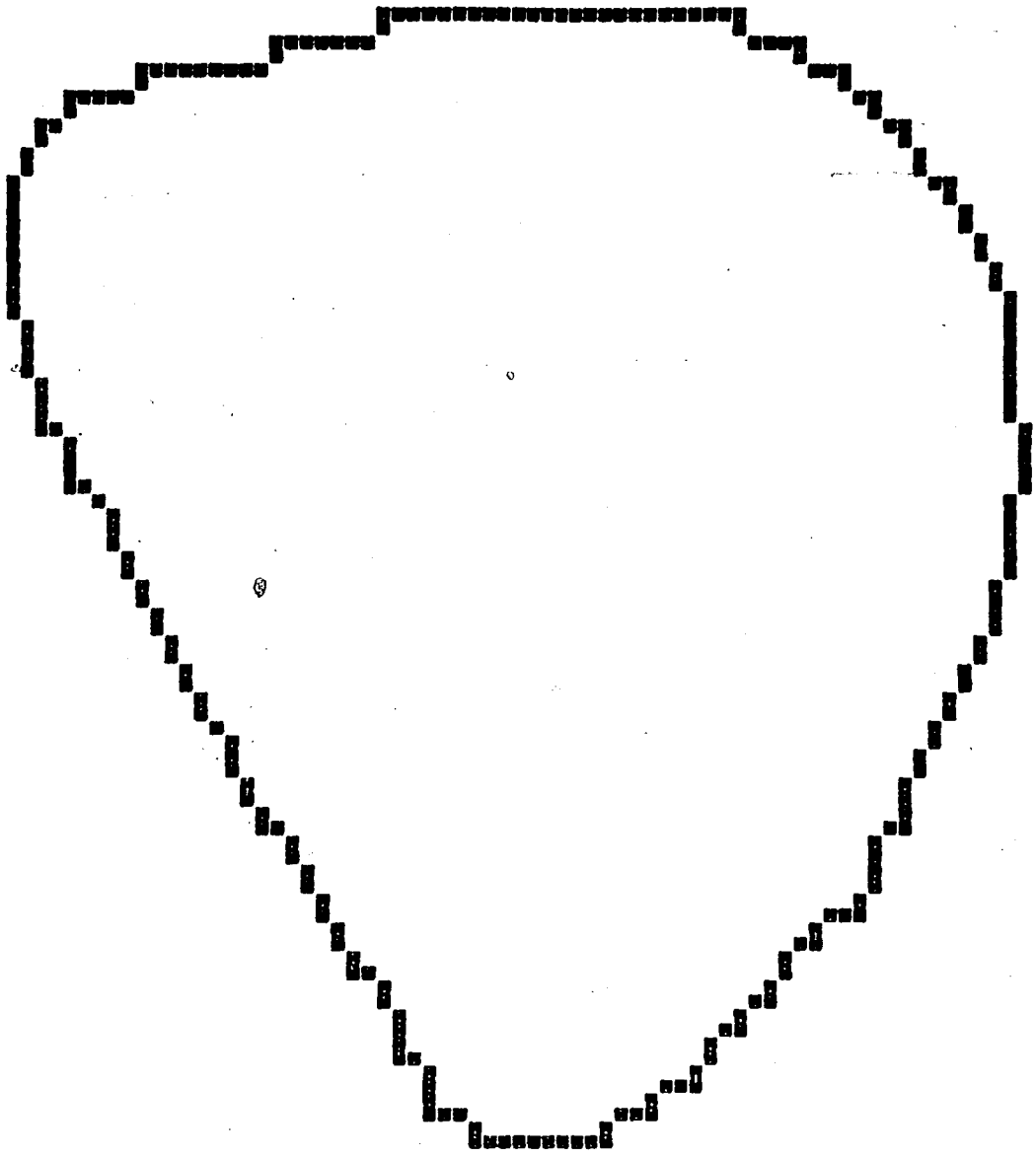


FIGURE 23

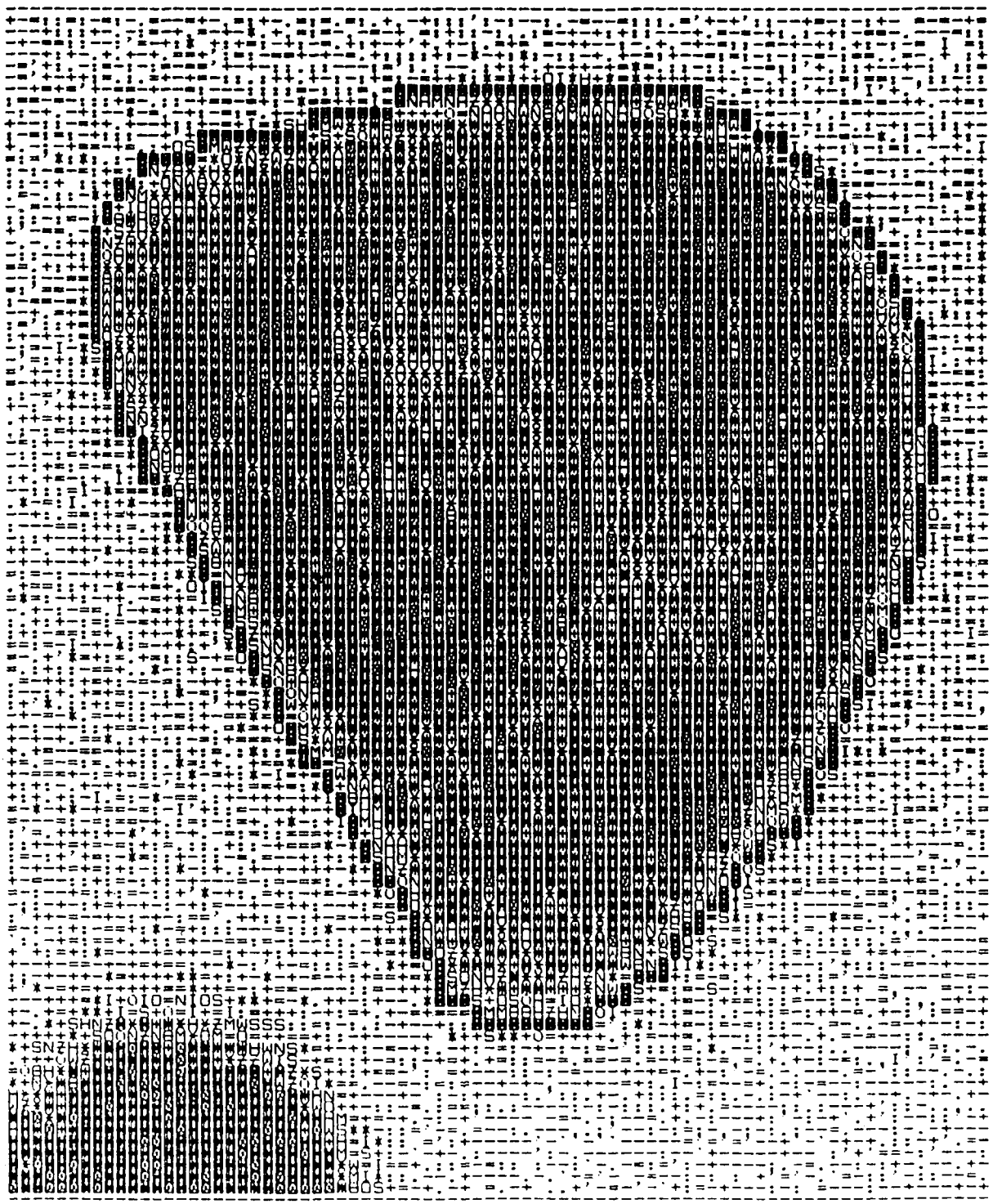


FIGURE 24

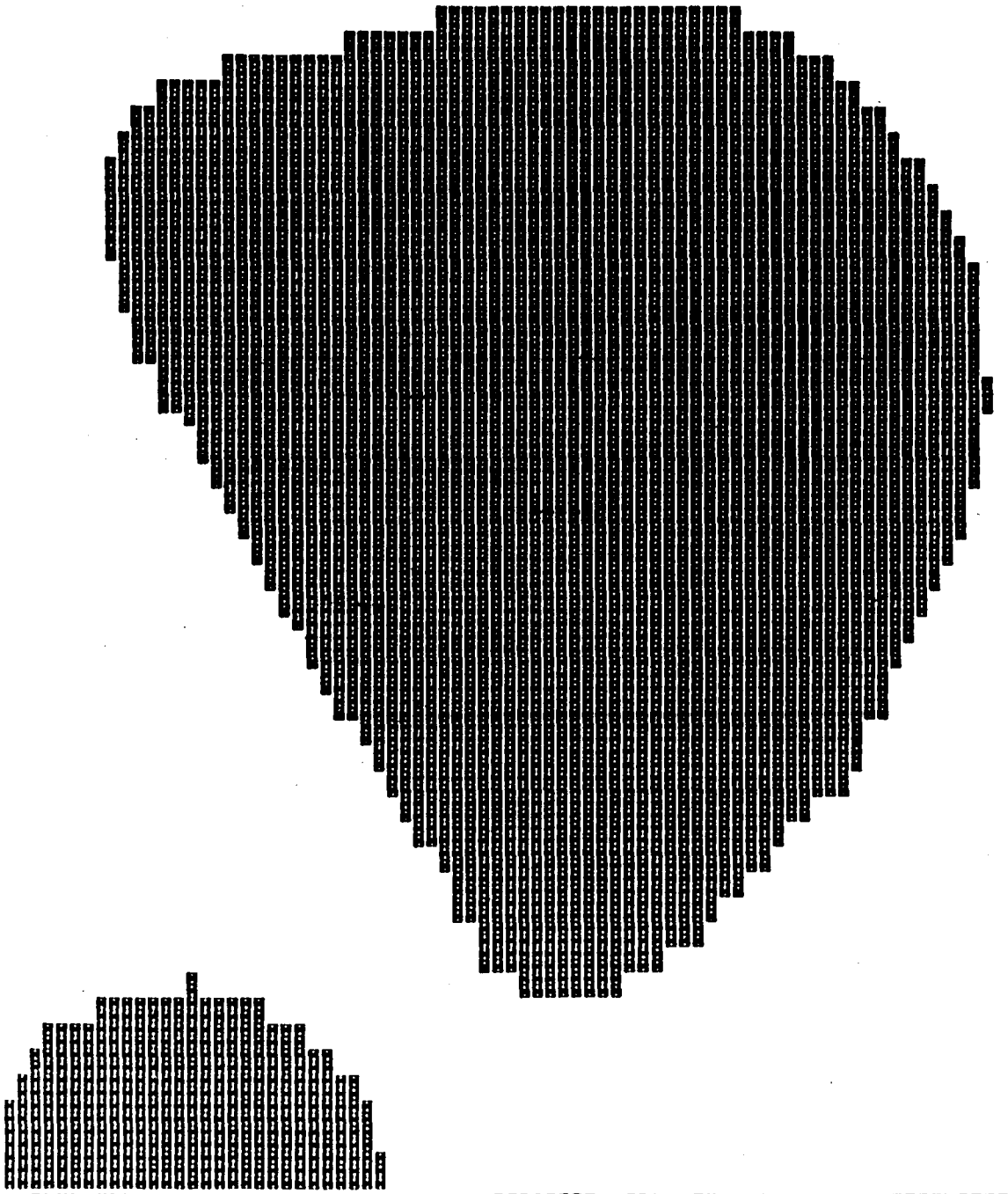


FIGURE 25

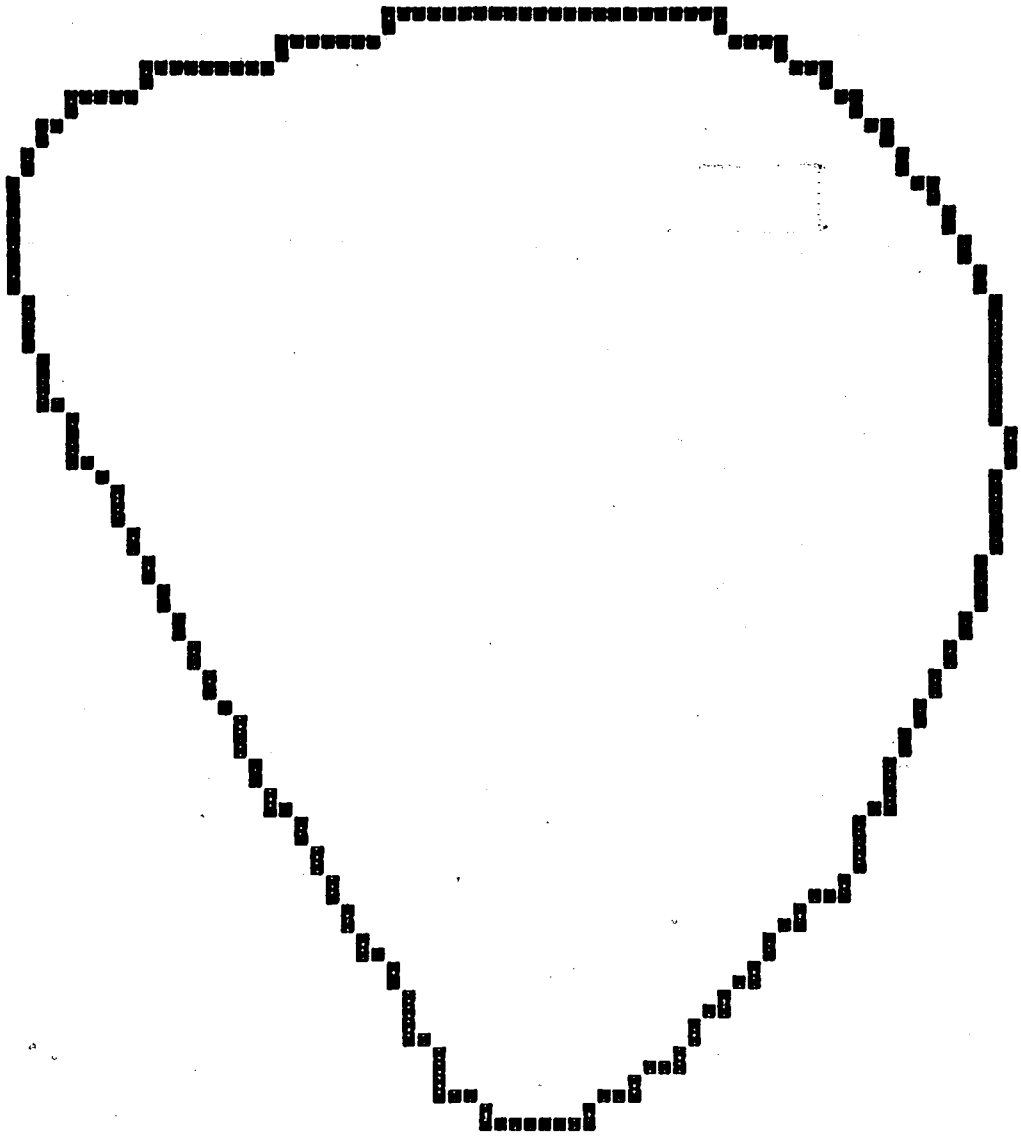


FIGURE 26

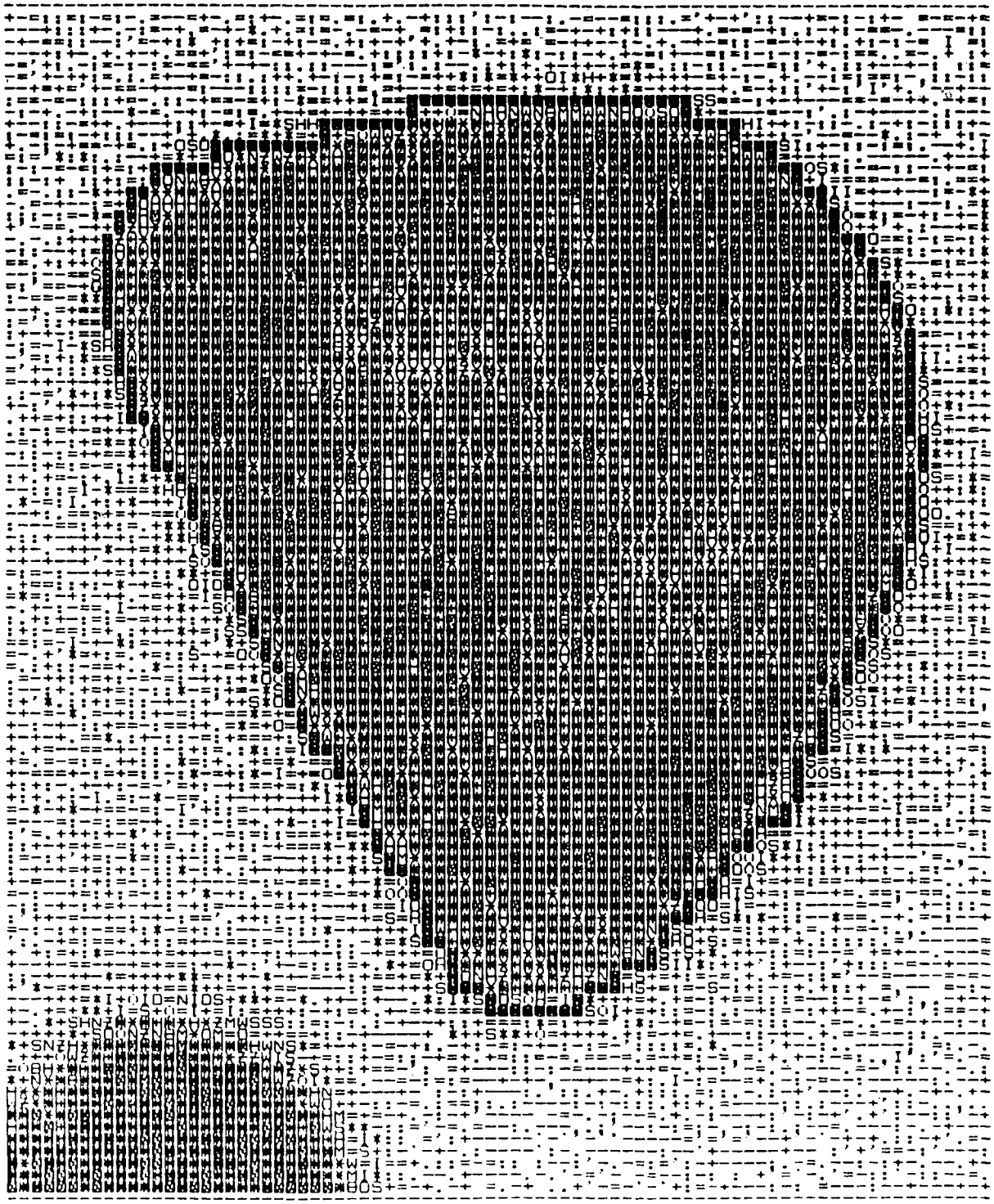


FIGURE 27

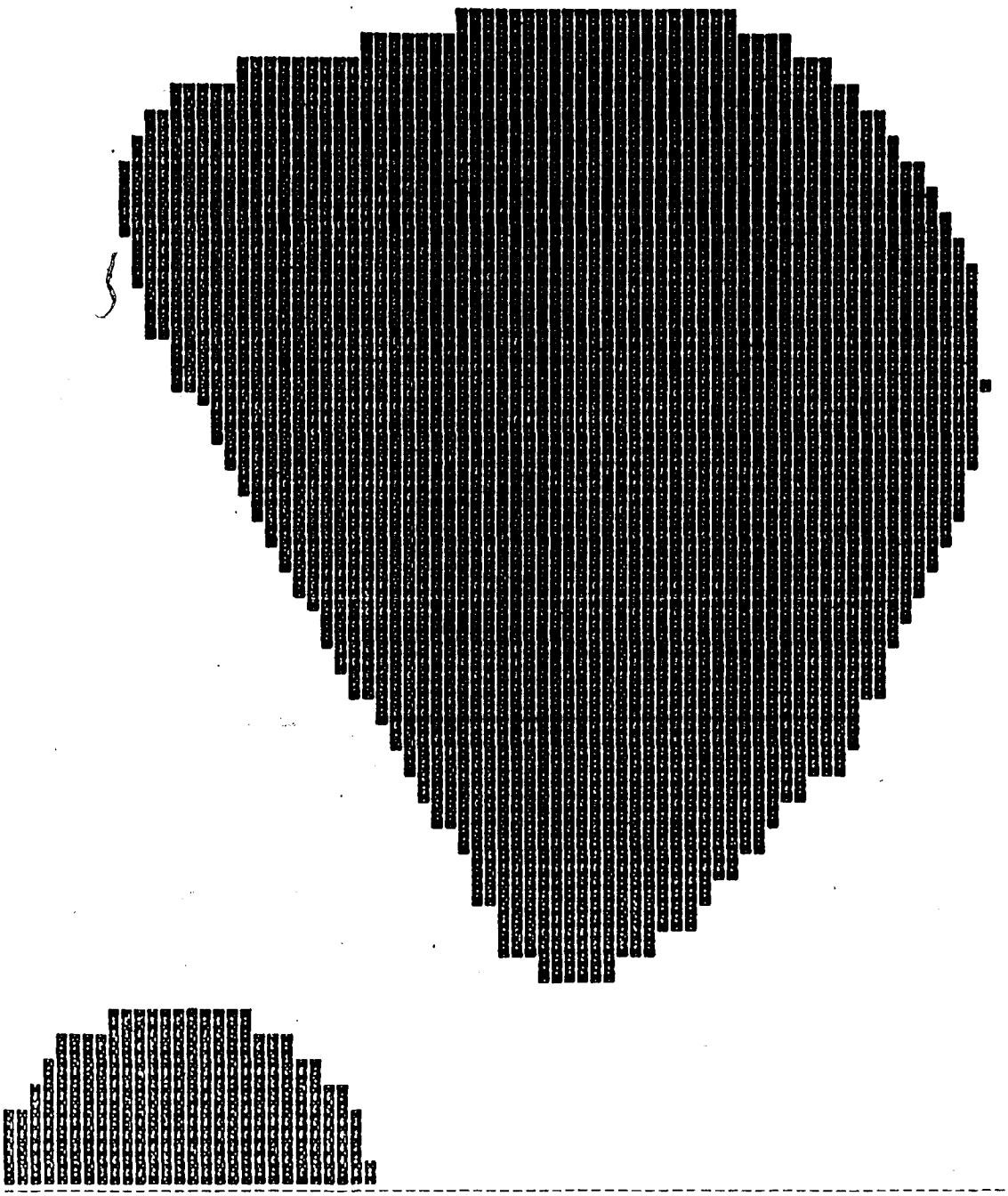


FIGURE 28

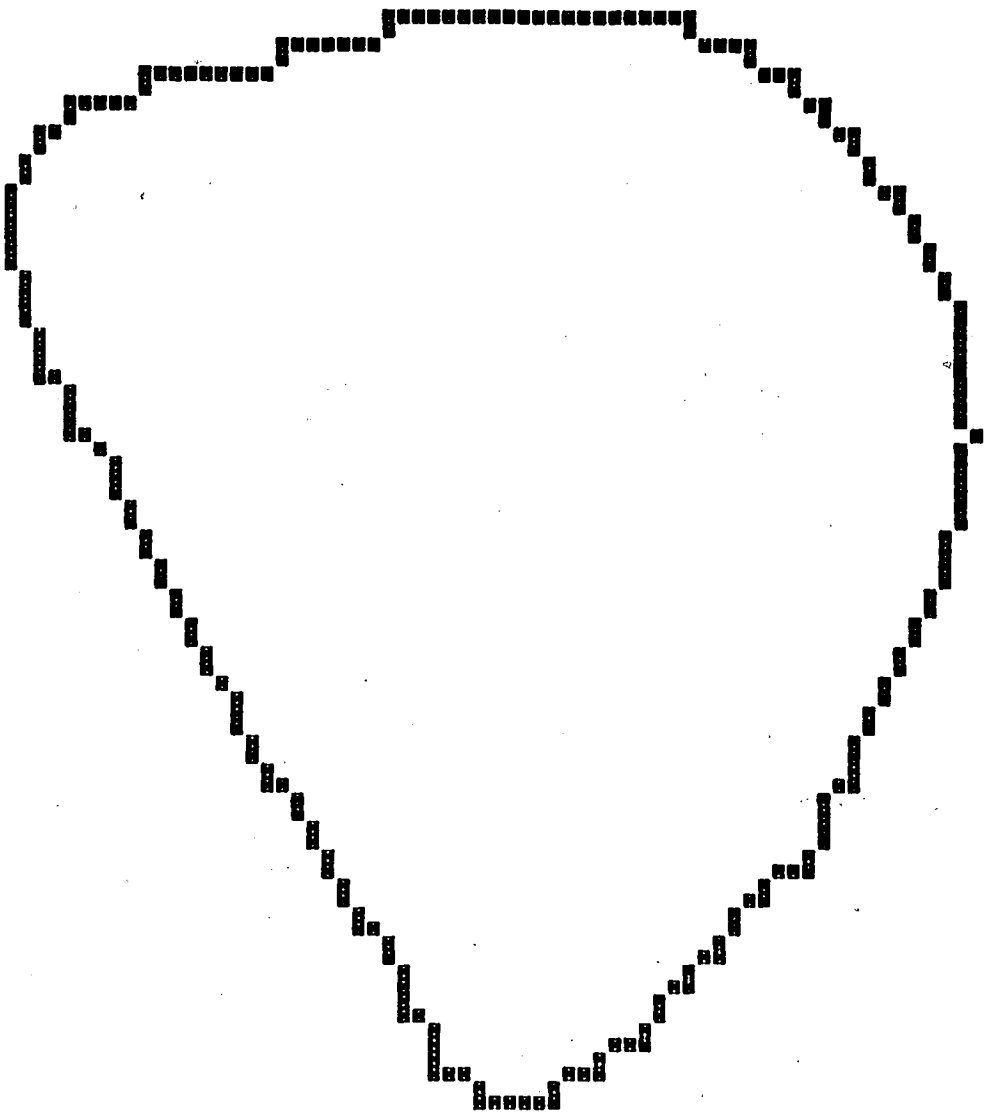


FIGURE 29

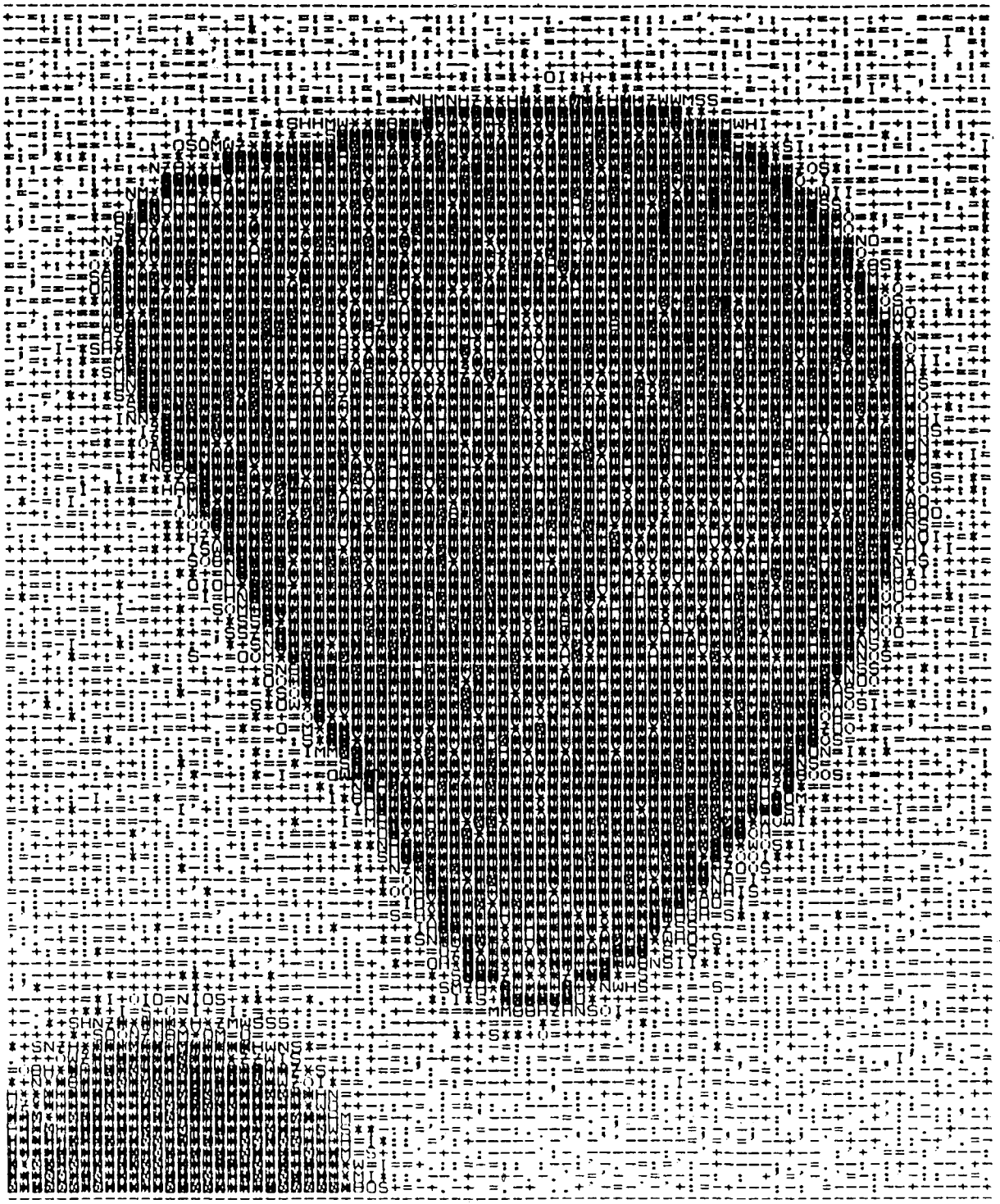


FIGURE 30

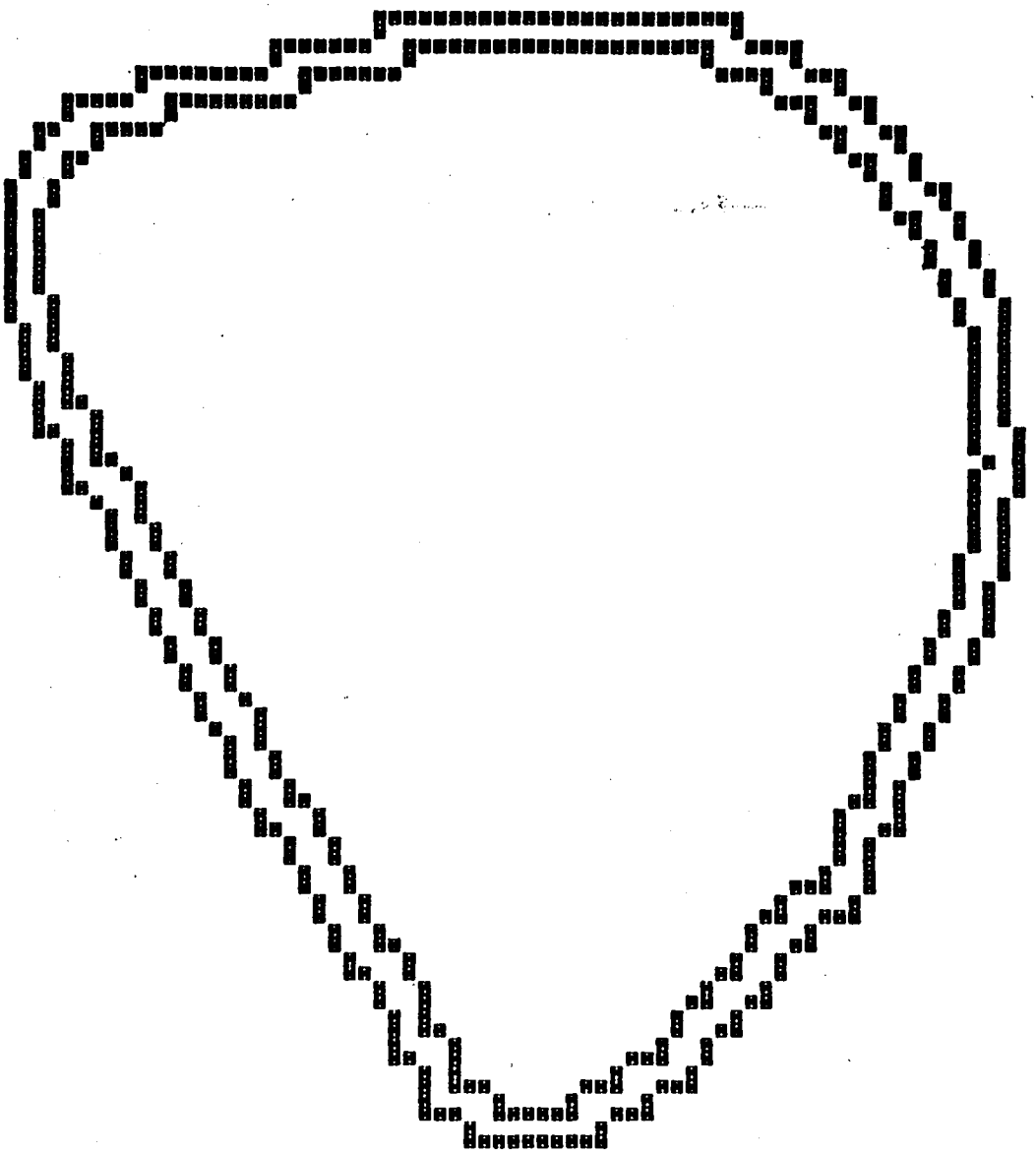


FIGURE 31

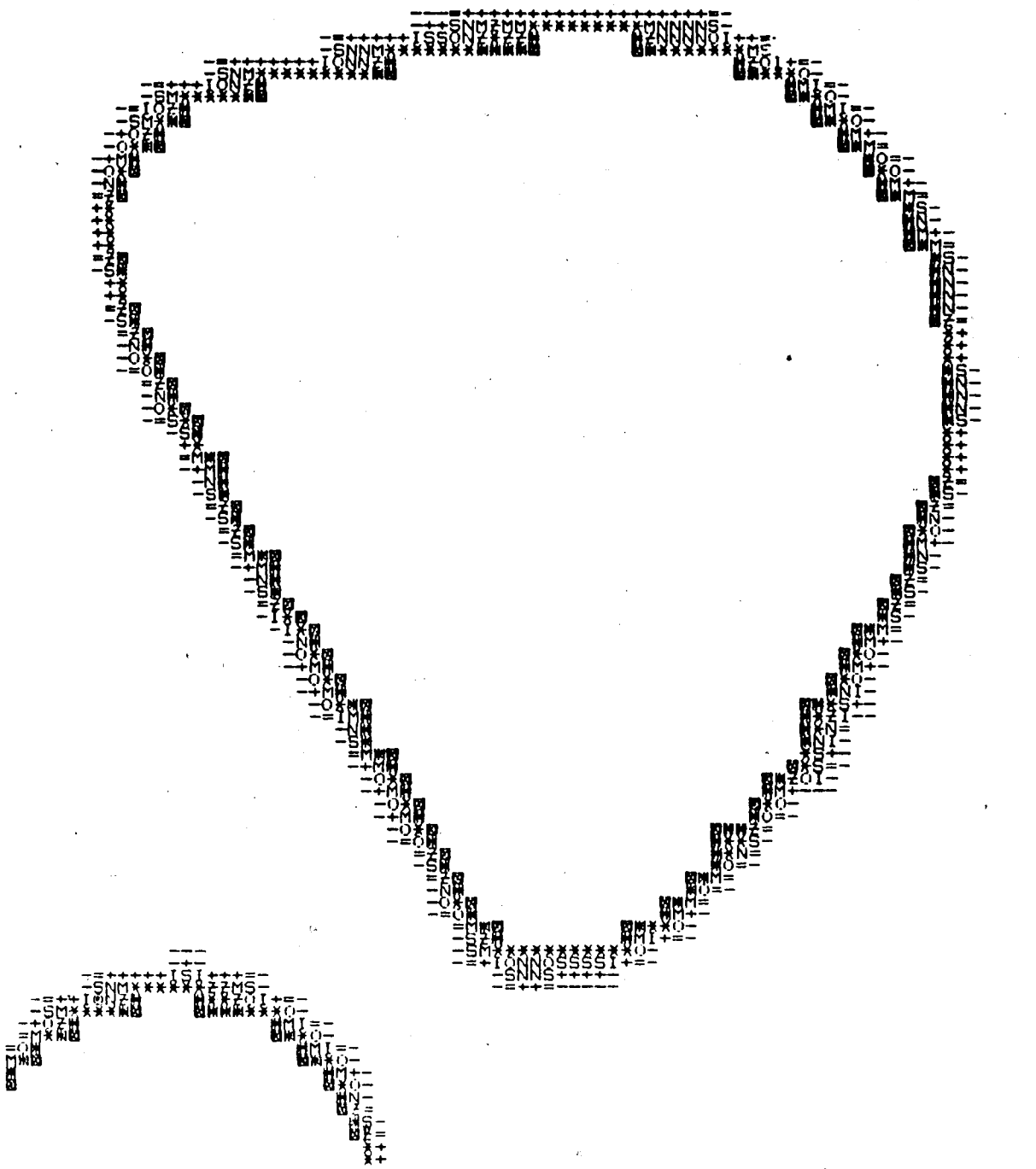


FIGURE 32

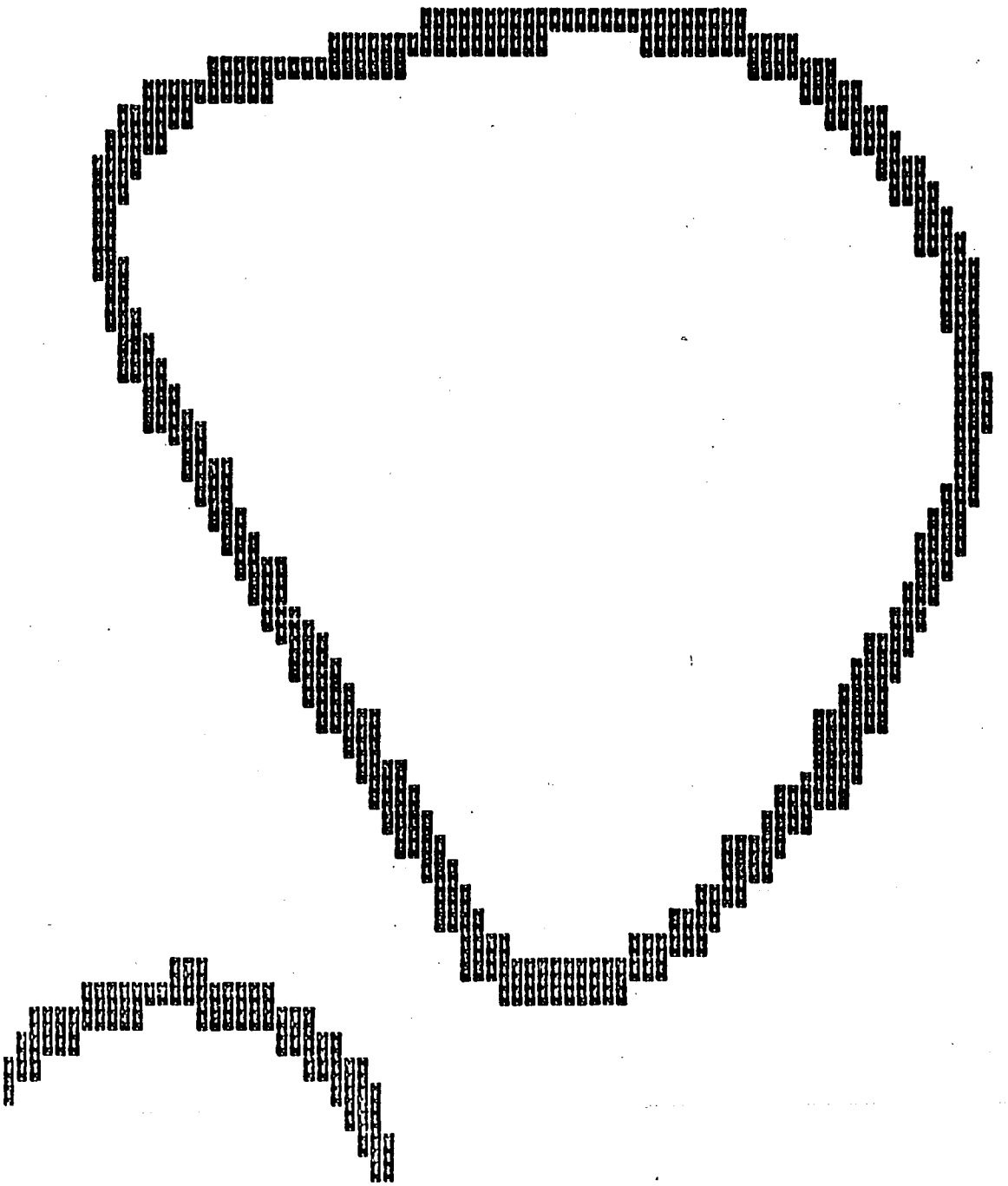


FIGURE 33

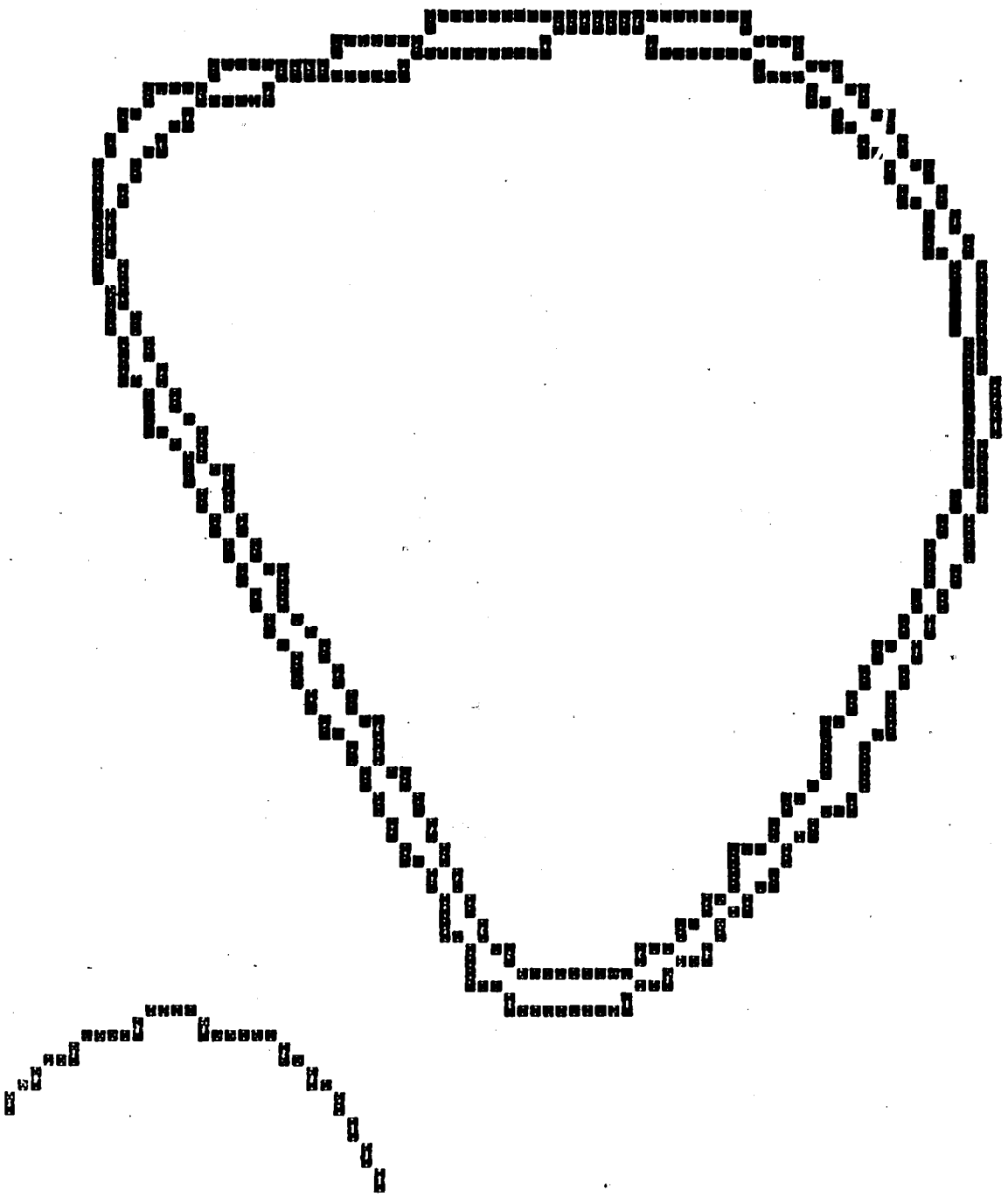


FIGURE 34

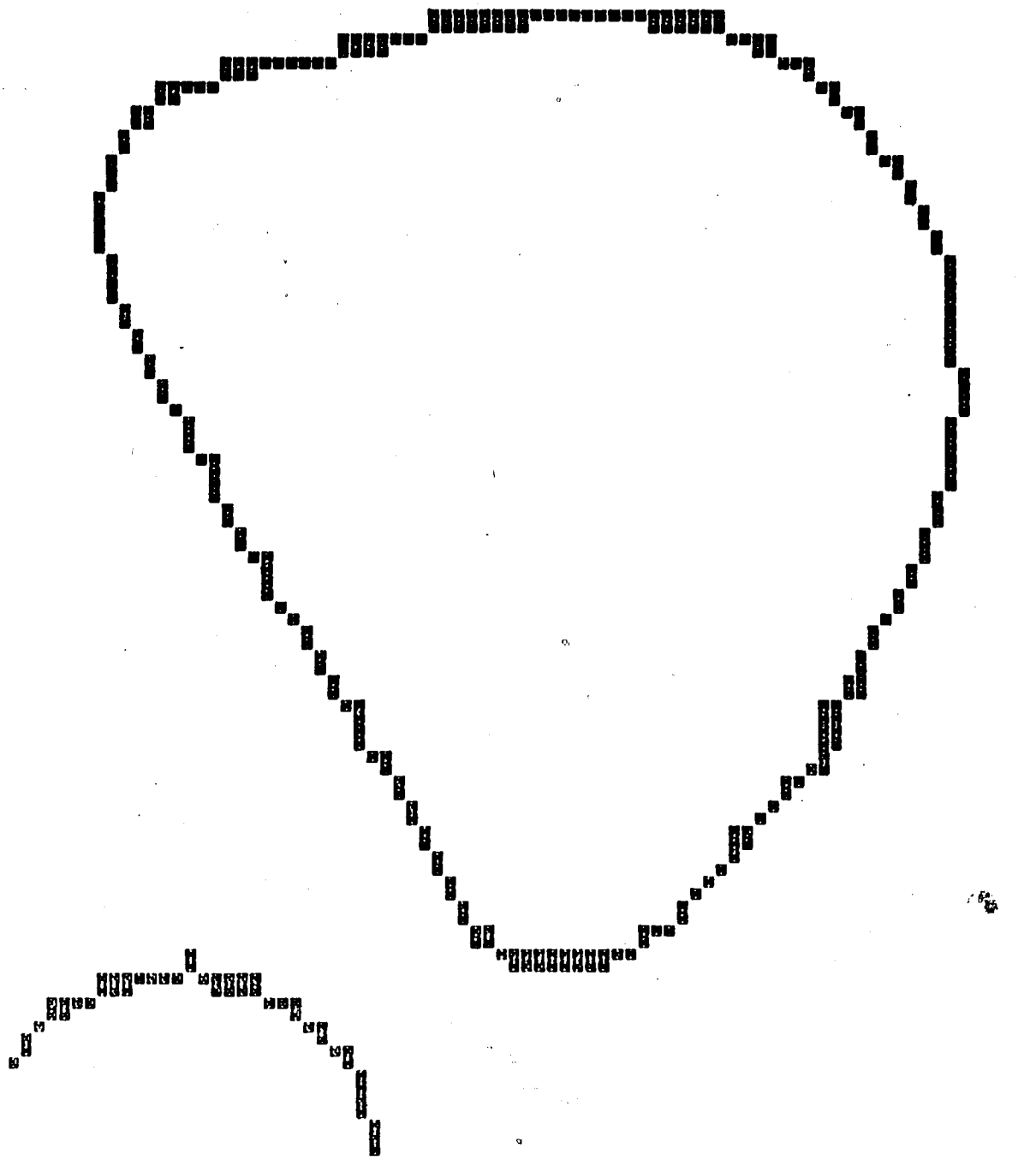


FIGURE 35

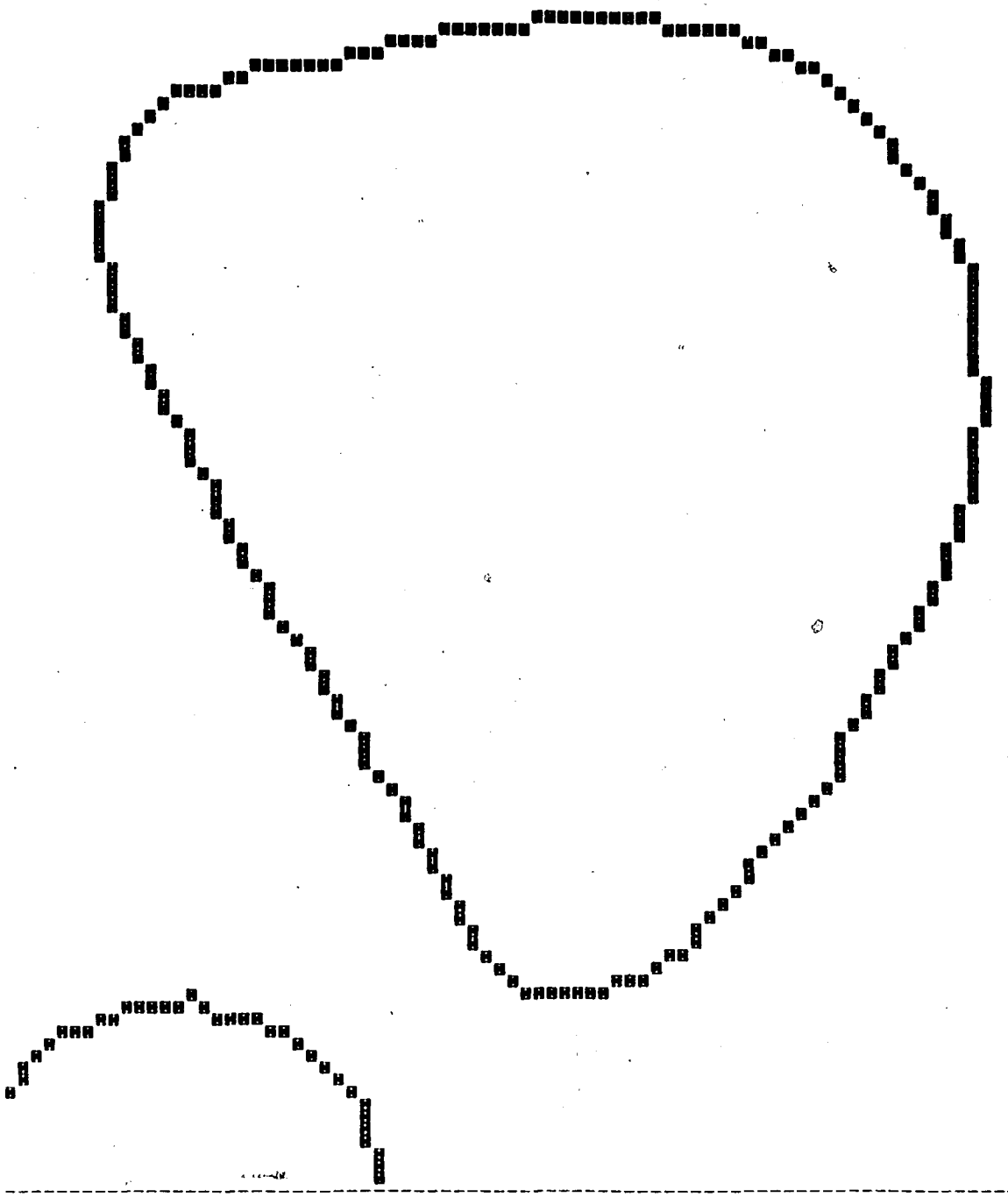


FIGURE 36

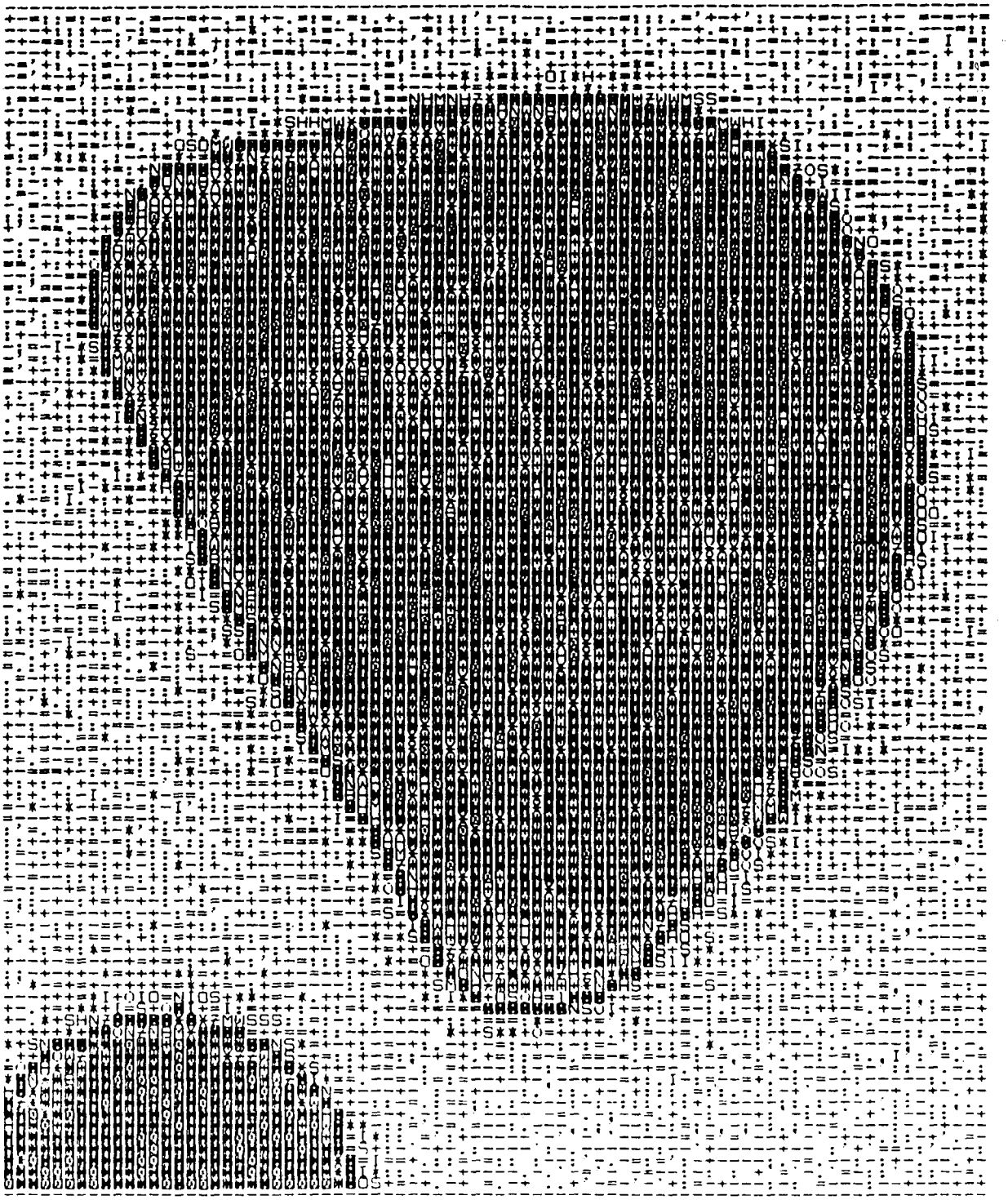


FIGURE 37

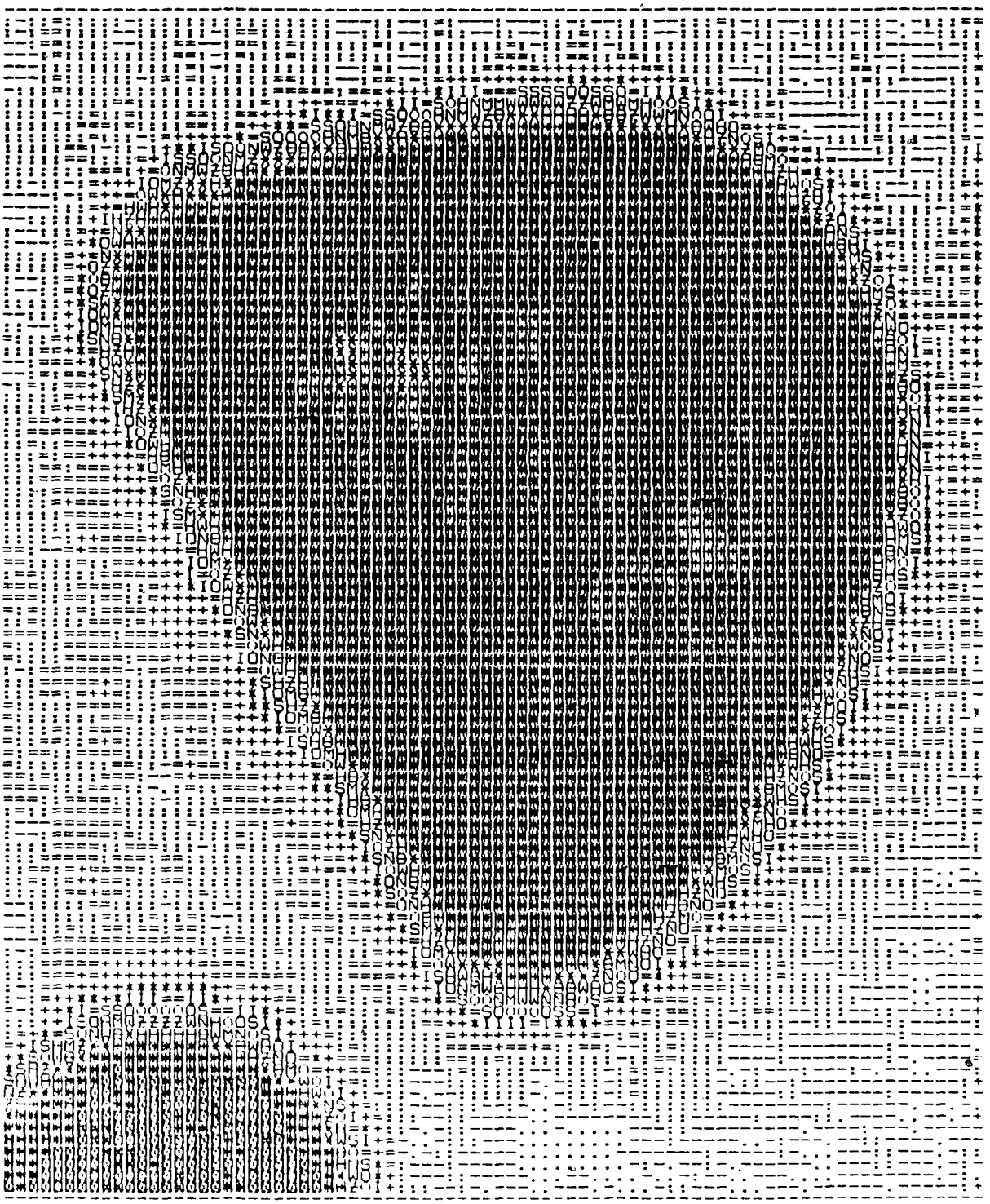


FIGURE 38

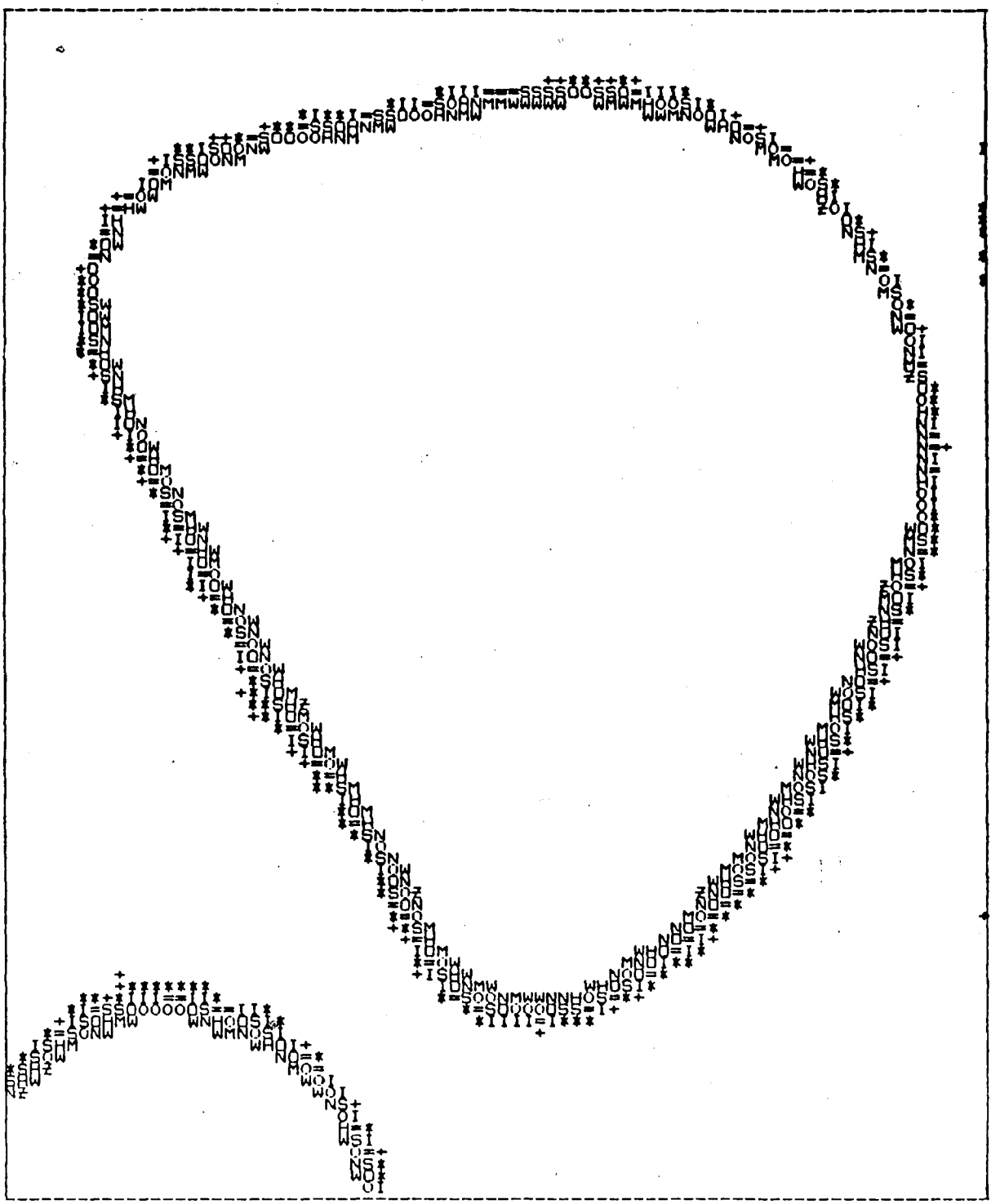


FIGURE 39

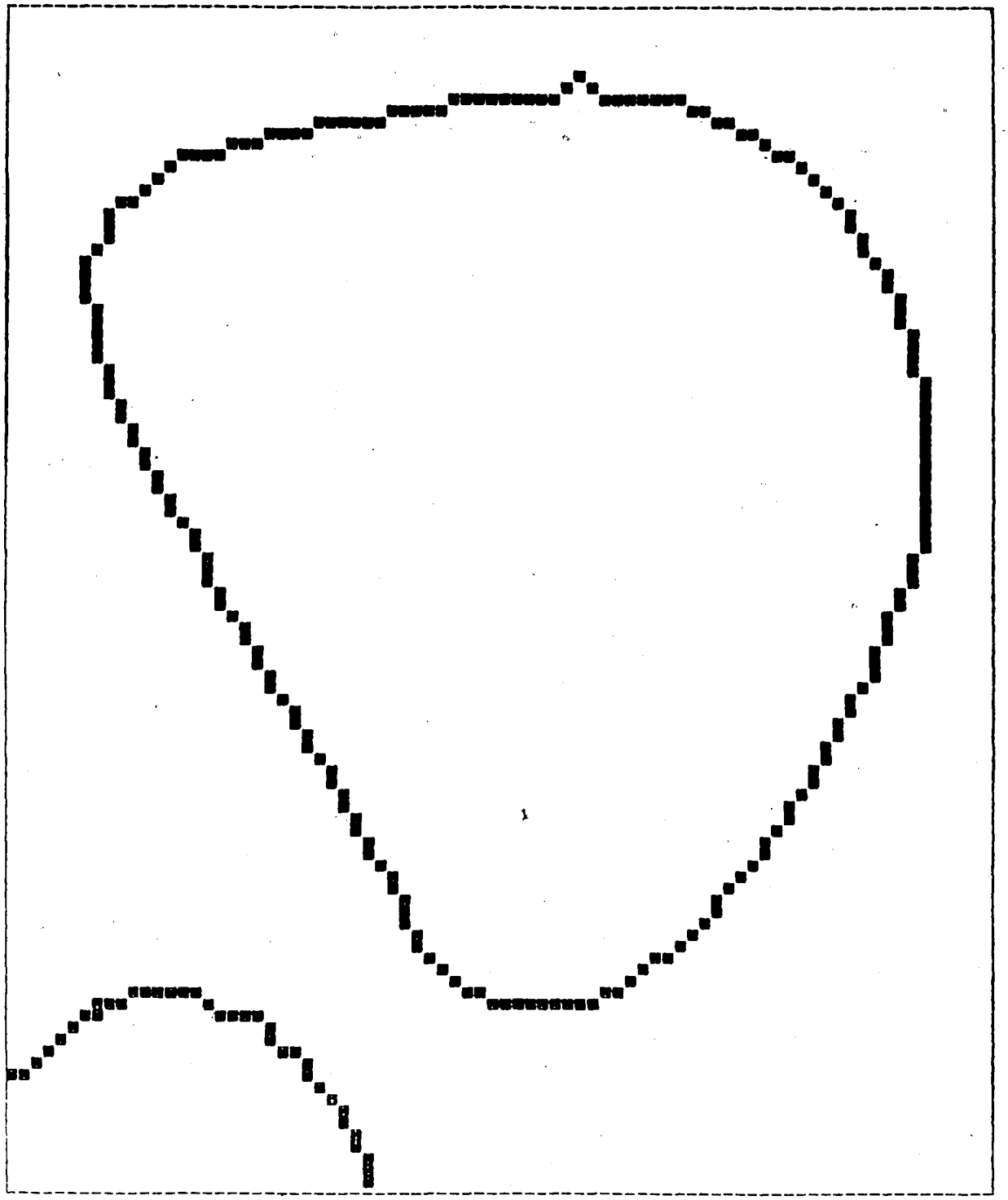


FIGURE 40

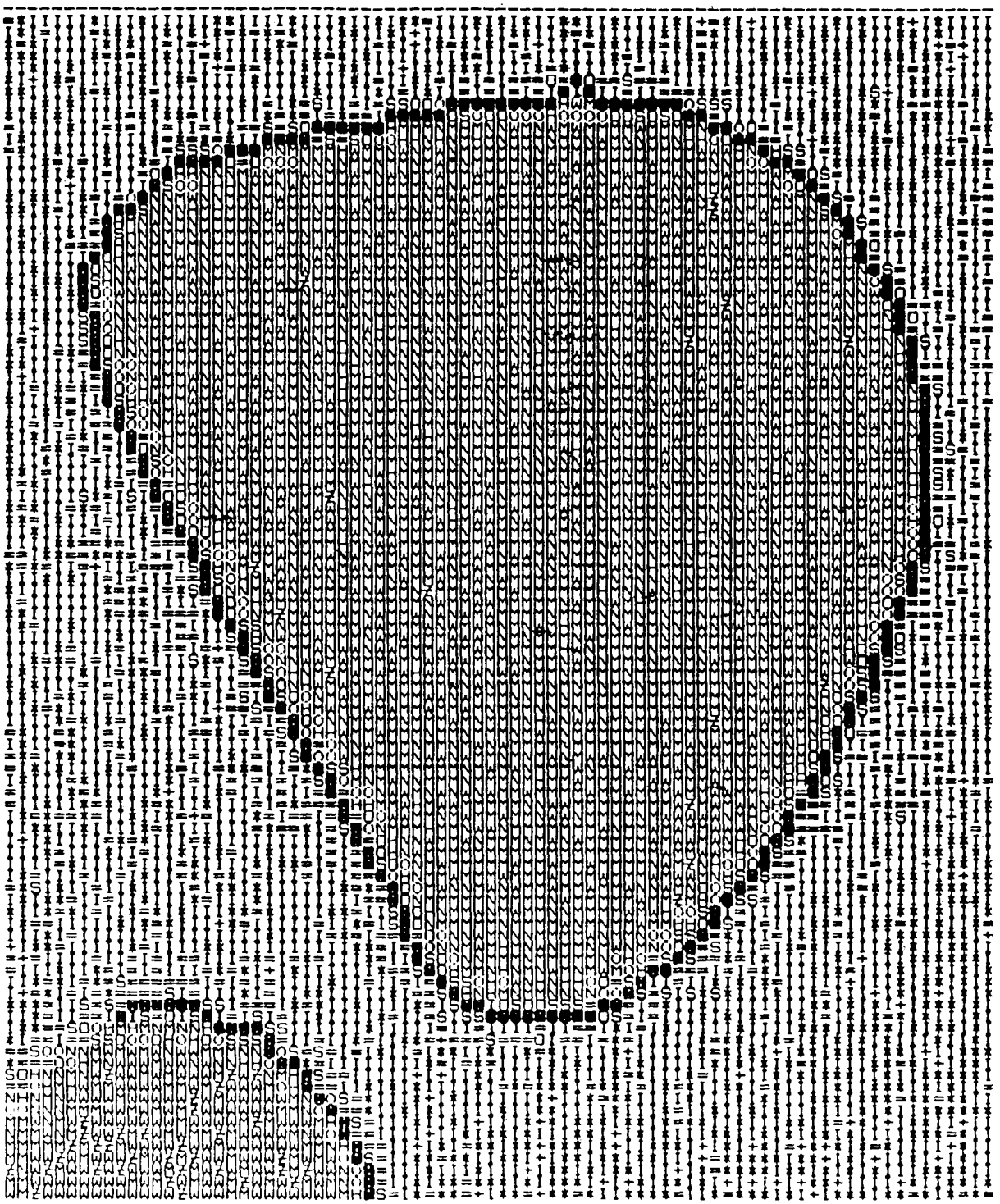


FIGURE 41

APPENDIX A

To configure the DGS800.MAC program for Microsoft Fortran and the IMS computer system, the beginning of the program should look as follows:

```
        MICROSOFT       1       FORTRAN
        MBASIC          Ø       BASIC INTERPERTER
        SORCIM          Ø       PASCAL
        MT MICRO        Ø       PASCAL MT+
        TWO MGHZ        Ø       TWO MHZ CLOCK SPEED
        FOUR MGHZ       1       FOUR MHZ CLOCK SPEED
        SIX MGHZ        Ø       SIX MHZ CLOCK SPEED

WSØ     EQU     AØ          CAT I/O BASE ADDRESS
SRAM    EQU     EØØØ        MEMORY BASE ADDRESS
MPWNDW  EQU     FØØØ        COLOR MAP WINDOW
FS      EQU     AC          I/O PORT FOR FRAME GRABBER
```

The program is now ready to be assembled, and can be assembled by using any availiable assembler. The assembler used by the author was M8Ø, and the assembly is completed by typing:

```
M8Ø =DGS8ØØ
```

This creates a file called DGS8ØØ.REL a relocatable file.

VITA

Paul F. Hemler was born in Lancaster Pennsylvania on November 16,1957. He was the the third child of Paul M. Hemler and Elizabeth B. Hemler. He attended Villanova University as an undergraduate student, and received a BSEE degree in May 1980. While at Villanova, he was elected to Eta Kappa Nu, the Electrical Engineering Honor Society. After graduation from Villanova he accecpted an engineering position with Western Electric. During his two and a half years at Western Electric, he worked closely with Bell Laboratories to design lightwave transmitters and receivers for high bit rate applications. He returned to attend Graduate School on a full time basis and accecpted a teaching assistantship at Lehigh University. He has taught several undergraduate classes and did research for the Geology Department while at Lehigh. He will receive an MSEE Degree in May 1984.