INFORMATION TO USERS

This reproduction was made from a copy of a manuscript sent to us for publication and microfilming. While the most advanced technology has been used to photograph and reproduce this manuscript, the quality of the reproduction is heavily dependent upon the quality of the material submitted. Pages in any manuscript may have indistinct print. In all cases the best available copy has been filmed.

The following explanation of techniques is provided to help clarify notations which may appear on this reproduction.

1. Manuscripts may not always be complete. When it is not possible to obtain missing pages, a note appears to indicate this.

2. When copyrighted materials are removed from the manuscript, a note appears to indicate this.

3. Oversize materials (maps, drawings, and charts) are photographed by sectioning the original, beginning at the upper left hand corner and continuing from left to right in equal sections with small overlaps. Each oversize page is also filmed as one exposure and is available, for an additional charge, as a standard 35mm slide or in black and white paper format. *

4. Most photographs reproduce acceptably on positive microfilm or microfiche but lack clarity on xerographic copies made from the microfilm. For an additional charge, all photographs are available in black and white standard 35mm slide format.*

*For more information about black and white slides or enlarged paper reproductions, please contact the Dissertations Customer Services Department.

UMI Dissertation Information Service

University Microfilms International
A Bell & Howell Information Company
300 N. Zeeb Road, Ann Arbor, Michigan 48106
Wu, Chris Kung

COMPUTER AIDED GRAIN SIZE DISTRIBUTION ANALYSIS OF POLYCRYSTALLINE MATERIALS

Rice University

University Microfilms International

300 N. Zeeb Road, Ann Arbor, MI 48106

M.S. 1986
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark ✓.

1. Glossy photographs or pages ______
2. Colored illustrations, paper or print ______
3. Photographs with dark background ✓
4. Illustrations are poor copy ______
5. Pages with black marks, not original copy ______
6. Print shows through as there is text on both sides of page ______
7. Indistinct, broken or small print on several pages ✓
8. Print exceeds margin requirements ______
9. Tightly bound copy with print lost in spine ______
10. Computer printout pages with indistinct print ______
11. Page(s) ______ lacking when material received, and not available from school or author.
12. Page(s) ______ seem to be missing in numbering only as text follows.
13. Two pages numbered ______. Text follows.
14. Curling and wrinkled pages ______
15. Dissertation contains pages with print at a slant, filmed as received ______
16. Other ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________

University
Microfilms
International
RICE UNIVERSITY

COMPUTER AIDED GRAIN SIZE DISTRIBUTION
ANALYSIS OF POLYCRYSTALLINE MATERIALS

by

CHRIS KUNG WU

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

MASTER OF SCIENCE

APPROVED, THESIS COMMITTEE

John B. Cheatham, Jr., Professor of Mechanical Engineering, Co-Director

Rui J. P. DeFigueiredo
Professor of Electrical and Computer Engineering, Co-Director

Franz R. Brotzen
Professor of Materials Science

Houston, Texas

April, 1986
Computer Aided Grain Size Distribution
Analysis of Polycrystalline Materials

Chris Kung Wu

Abstract
This thesis describes a semi-automated, computer-aided, digital image analyzer for analyzing the grain size distribution of polycrystalline materials. The system used in this work consists of an electronic micrograph scope, a black and white television camera, an image digitizer, and a personal computer. Individual grain sizes in the structure are obtained from a digitized image of a grain structure using edge detection, histogram segmentation, and label-merge techniques.

A synthetic grain structure drawing and a micrograph photograph of an aluminum specimen are examined to test the system. Comparisons made between the mean size measured by the system and that by a planimeter shown that the grain size image analyzer produces surprisingly good correlations. When the image analyzer is used in a fully automatic mode, the discrepancy between the average areas computed by the analyzer and measured by the planimeter are approximately 20 percent; however, with operator assistance the discrepancy is reduced to 7 percent or less.

A discussion of digital image processing techniques is presented for better technical understanding of the algorithms used in this system. Experimental results and the software implementations of these techniques are included in Appendix A and D.
Acknowledgements

I extend my gratitude to my principal adviser Dr. John B. Cheatham for all the assistance he has given me throughout the period I worked on my thesis; to the Co-Director Dr. Rui J. P. DeFigueiredo for his support and technical advice; to Dr. Franz R. Brotzen for his assistance in this effort; and to Joe Gesenhues for his help in the laboratory.

Additionally, I greatly appreciate the support and advice from the Tracking and Communications Division of the NASA Johnson Space Center.

Finally, a special thanks goes to my family, who has been patient and supportive through the years.
Table of Contents

<table>
<thead>
<tr>
<th>I</th>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-1</td>
<td>Preliminary Remarks</td>
</tr>
<tr>
<td>I-2</td>
<td>Problem Definition</td>
</tr>
<tr>
<td>II</td>
<td>System Description</td>
</tr>
<tr>
<td>II-1</td>
<td>Data Acquisition</td>
</tr>
<tr>
<td>II-2</td>
<td>Histogram Segmentation</td>
</tr>
<tr>
<td>II-3</td>
<td>Wireframe Extraction (Edge Detection)</td>
</tr>
<tr>
<td>II-4</td>
<td>Label-Merge Algorithm for Grain Size Measurement</td>
</tr>
<tr>
<td>II-5</td>
<td>Least-Square Curve Fitting</td>
</tr>
<tr>
<td>III</td>
<td>Experimental Results</td>
</tr>
<tr>
<td>IV</td>
<td>Concluding Remarks</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
</tr>
<tr>
<td>Appendix A:</td>
<td>Digital Image Processing</td>
</tr>
<tr>
<td>A-1</td>
<td>Sampling and Quantization</td>
</tr>
<tr>
<td>A-2</td>
<td>Noise Measurement</td>
</tr>
<tr>
<td>A-3</td>
<td>Edge Enhancement</td>
</tr>
<tr>
<td>A-4</td>
<td>Noise Removal</td>
</tr>
<tr>
<td>A-5</td>
<td>Edge Detection</td>
</tr>
<tr>
<td>A-6</td>
<td>Summary</td>
</tr>
<tr>
<td>Appendix B:</td>
<td>Introduction to Chorus &quot;PC1000&quot; 6-bit Digitizer and IBM-PC Memory Management</td>
</tr>
<tr>
<td>Appendix C:</td>
<td>Raw Data</td>
</tr>
<tr>
<td>Appendix D:</td>
<td>Program Listing</td>
</tr>
</tbody>
</table>
I. Introduction

I-1 Preliminary remarks

Measuring techniques in quantitative metallography are one of the important aspects in the study of the internal structure of polycrystalline materials. In general, the measuring techniques can be categorized into the following operations:

1. Counting
2. Size determination
3. Area fraction measurement

A survey of the quantitative measurement techniques is provided by Fischmeister [12], and Exner [13]. These techniques are mostly approximation types of measurements. Some of the techniques depend on the assumption that the particles in the grain structure closely approach a spherical shape. Some of them employ electronic devices for analog data processing. These devices provide fairly accurate results, but are expensive and the measurements are specialized for some specific materials. A knowledge of the grain size distribution is often required in the study of metallurgy. This thesis provides a general purpose computerized grain structure image analysis system for measuring the size of each grain. There are no assumptions regarding the shape of the grain and a complete grain size distribution curve is produced. This system uses a picture of the grain structure for input. Using a personal computer, the picture is transformed into a wireframe picture and the grain size is measured using the label-merge algorithm from the wireframe picture. A wireframe picture is a
Figure 1 Digital image analyzer system flow chart
picture which contains only the edges of the objects in the picture. An accumulated grain size distribution curve is plotted as a function of the area. The least square method is adopted for fitting a smooth curve from which the distribution density function and its statistics are derived. Figure 1 shows the system flow chart. This computerized system provides not only semi-automatic measurement but also reasonable results. In the following, section 2 provides a description of the system in detail. Section 3 includes a discussion of the results produced from this system and a comparison of the results from measurements with a planimeter. Chapter 4 describes the merits of this system. Appendix A provides a comprehensive study on the image processing techniques. Appendix B includes the software implementation of the algorithms used in the system and techniques discussed in Appendix A. Finally, Appendix C includes the raw data produced from the system.

1-2 Problem Definition

Most of the common materials of engineering are polycrystalline materials. These materials consist of a large number of crystals bonded to one another along interfaces called grain boundaries. Figure 2 shows an example of the grain structure in aluminum.

Grain size has an important effect on diffusion coefficients, mechanical properties, electrical and magnetic properties, and phase diagrams. In metallurgical literature, the material properties are sometimes expressed as functions of the grain size. For this reason
Figure 2  Micrograph of an aluminum specimen (x100)

Figure 3  System configuration
the control of grain size and the measurement of the size are important considerations in Metallurgy.

The grain size distribution of a polycrystalline material can be expressed as a function of volume, cross-section area, or any other parameter reflecting the grain size [11]. In some metallurgical research laboratories, sophisticated devices are used for grain size measurements. Such equipment is highly accurate but very expensive. This high cost makes it unavailable to most researchers. Alternatively, grain size can be measured manually with a planimeter on the photomicrograph of the grain structure. This method provides a reliable measurement but it is not practical when a large number of grains are present in the photomicrograph. To overcome this inefficiency problem, other methods such as counting the grains within a unit area or measurement of the average grain diameter are adopted for crude measurements. These alternatives can be easily performed but they provide little information regarding to the grain size distribution. A computerized system designed for measuring the grain size distribution is described in this thesis. This system employs the image processing techniques discussed in Appendix A. Figure 3 shows the configuration of the system used in the experiments. This system consists of:

- A Reichert metallograph scope.
- An RCA black and white T.V. camera mounted on the micrograph is used to take the image of the photomicrograph of the grain structure and transmit the image to a digitizer.
- A Chorus 6-bit digitizer. This computer controlled image digitizer quantizes the analog image transmitted from the T.V. camera into a rectangular array of integers (digital image) and stores the array in the computer.
- A digital image processing unit. This software implemented unit retrieves the digital picture and generates a wireframe representation of it.
- A software implemented label-merge algorithm designed for grain-size counting. A list of the grain size distribution as well as its statistics is output from this program.
- An IBM Personal Computer with 640K-bytes of Memory and the control program.

II. System Description

In section I, we have discussed the framework of the proposed system for grain size distribution analysis. The following sections introduce the algorithms used in this system.

II-1 Data Acquisition

The picture used for the grain size distribution analysis is taken with a TV camera from a specimen surface through a micrograph. The output analog picture from the TV camera is quantized into a digital picture with 64 levels of gray. A full discussion on sampling and quantization process is provided in Appendix A-1. Appendix B provides an introduction to the digitizer used in this system and the
frame memory management in the IBM PC. Figures 4 and 5 show the analog and the digitized pictures, respectively.

The graphics card (TECMAR Graphics Master) used in the system provides a resolution of 640 x 400 pixels with 16 colors. The true gray shades can not be displayed using this graphics card. Due to this limitation, the digital picture is mapped into a color picture for displaying. The mapping from gray shades to color is performed using a mapping table. Optimally, this mapping table is set up so that the difference in wavelengths of two adjacent colors is maximum. This mapping table is usually set up internally by the manufacturer. There are 64 levels of gray present in the digital picture and a 16 color display capability. This disagreement prevents a one to one mapping so that every 4 adjacent shades are mapped into the same color.

From the digital picture shown in Figure 5, several problems associated with the quantized picture are revealed:

(a) Unevenly distributed light intensity

This is shown from the disagreement of light intensity between the left and right half of the picture in Figure 5. This discrepancy can be lessened by applying the operation of Histogram equalization [A-3-1] on the digital picture.

(b) False edges

For two adjacent grains with close gray levels, it is questionable that they are truly separate grains as shown or a single grain with a false edge in between them.
Figure 4  Grains sampled from Figure 2 (x390)

Figure 5  Result of quantization on Figure 4 with Chorus 5-bit digitizer
Figure 6  Model of grain structure

Figure 7  Result of quantization on Figure 6 with
           Chorus 6-bit digitizer
-X- GrayLevels, -Y- Frequency of Occurrence.

Figure 8  Histogram of Figure 7

Figure 9  (a) Sample from Figure 7  
(b) Result of histogram segmentation
-X- GRAYLEVELS, -Y- FREQUENCY OF OCCURRENCE

Figure 10  Histogram of Figure 5

(a)  (b)

Figure 11  (a) Sample from Figure 5
            (b) Result of Histogram Segmentation into
                two regions.
            (The band shown in the picture resulted from the
             author's poor photographic techniques)
(c) Noise

Inspecting each grain contained in Figure 5, we see that there are several graylevels present in a grain. This raises difficulty in wireframe extraction.

Section II-2 provides a discussion of an iteration process using a histogram segmentation technique for solving the false edge and noise problems.

The digitized picture is an array of 640 x 400 integers (512 K). Since it is too large to be processed with a personal computer of 640 K memory, the picture is subdivided into subpictures of size 256 x 200 and each of these is processed separately.

II-2 Histogram Segmentation

In order to understand the general characteristics associated with the false edge and noise problems, the following experiments were conducted. Figure 6 shows a model of the sampled grain structure in which only three levels of gray are present, and Figure 7 shows the digitized picture. A histogram of the image in Figure 7 is shown in Figure 8. The histogram of an image is a graph showing the number of occurrences of a graylevel in the image. Ideally, the histogram of the gray structure model would have only three spikes. Inspecting Figure 8, we see three peaks and some smaller spike diverging from them. This suggests that the false edges and noise are not randomly distributed but are localized. This observation implies that the picture can be restored by grouping the graylevels into several
segments and assigning new graylevels to each segment. This method is known as Histogram Segmentation. There are other techniques such as clustering and region growing that can be used for the segmentation process. However, the clustering techniques are designed for objects with known shapes and the region growing technique required a large amount of memory and processing time. The histogram segmentation is a process based on the global information obtained from the histogram analysis. It has the advantage of fast computation and less memory required.

For the picture of the grain structure model, it is not difficult to determine the range of graylevels for each segment. Figure 9 shows the result of segmenting the histogram into 3 regions with gray levels range from 0-35, 36-53, and 53-63, respectively.

For the case where the regions are not clearly separated as shown in Figure 10, an iteration searching process will be used for determining the segments. The searching algorithm searches the segments one by one and the operator determines the end of the search. The searching algorithm is described below:

(a) Smooth the histogram with a 3 point unweighted window. Scan through the smoothed histogram and mark the local minimums (region mark candidates) in the order from the lowest to the highest graylevel. In the previous case, the program selects 25, 30, 35 and 41 as the region mark candidates. See Figure C-2 in Appendix C. The selections are based on the following criterions:
1. It is a local minimum.

2. The distance between the selected marks have to be at least 4 graylevels apart.

3. At most 4 region marks can be selected. This number is chosen based on the apriori knowledge of the processed material.

(b) Use the first local minimum as the region mark. Segment the histogram into two regions and display the result.

(c) If the result is unsatisfactory, increase or decrease the region mark in the direction specified by the operator. Repeat step (b) using the adjusted region mark. If the result is satisfactory, fix the region mark and proceed to the next step.

(d) Segment the histogram into 3 regions using the first region mark found in the previous steps and select the next local minimum marked in step (a) as the second region mark.

(e) Readjust the second region mark as described in step (c) until a satisfactory result is achieved.

(f) Similar searching operations are carried out for determining the rest of the region marks. In the previous case, region marks are adjusted to 23, 29, 34, and 36 respectively.

Figure 11 and 12 show the intermediate results of the iteration searching process for Figure 5. Figure 13 shows a comparison of the original digital and the restored picture.
Figure 12  (a) Sample from Figure 5
(b) Result of Histogram Segmentation into three regions

Figure 13  (a) Sample from Figure 5
(b) Result of Histogram Segmentation into five regions
Figure 14  
(a) Sample from Figure 9(b)  
(b) Edges detected
Figure 15 Comparison between the grain sizes measured by planimeter and the sizes detected by the edge detector

- ▲ SIZE MEASURED WITH PLANI METER
- □ SIZE DETECTED BY EDGE DETECTOR
Figure 16  (a) Histogram Segmentation result
(b) Edges detected

Figure 17  Grains sampled from Figure 2 (x390)
110110101
110110101
000000111
110110101
110100000
110101111
000001111
111101111

(a)

110220303
110220303
000000333
440550303
440500000
440506666
000006666
777706666

(b)

Figure 18  (a) Wireframe model in which 0's represent the edges 1's represent the grain areas  (b) Wireframe data file after labeling
II-3 Wireframe Extraction (Edge Detection)

The edge detection is performed using a first derivative edge detector. Appendix A-5-1 gives a full description on this operator. Figure 14b shows the wireframe representation of the picture of the grain structure model shown in Figure 14a.

In order to check the accuracy of the output from the edge detector, Figure 15 shows a comparison between the size measured with a planimeter from the original picture shown in Figure 6 and the detected grain size. The raw data are contained in Appendix C. Statistical analysis shows that the difference between the average size measured by both methods is less than 2%. Figure 16 shows the wireframe representation of Figure 4.

II-4 Label-Merge Algorithm for Grain Size Measuring

Once the wireframe representation of a grain structure is generated, the next step is to determine the area of the individual grain. Table C-8 in Appendix C shows a portion of the wireframe data file of Figure 14 (b). The area of a grain in the digital wireframe picture can be defined as the number of points in the closed boundary representing the grain. With this definition, the measurement of the grain size reduces to a point counting process. The algorithm developed for the counting process is called the "Label-Merge" algorithm. Considering the model of the wireframe picture as shown in Figure 18 (a) in which 0's represent the edges and 1's represent the grain areas. The following procedure explains how this algorithm counts the areas in this model.
1. Extract the first row of data and label each segment between edge points with a non-zero number as depicted below.

110110101 raw data (first row)
110220304 labeled data (first row after labeling)

2. Count the number of points in each labeled segment and store in an area array as shown below. The labels are stored in an array indicating the boundary grains.

<table>
<thead>
<tr>
<th>label</th>
<th>number of points</th>
<th>boundary grain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

3. Extract the second row of data. Compare each of the segments in this row with the previous labeled row. If there are any points above this segment labeled, then label this segment with the same number. If none of the points above this segment has a label, then assign an unused label to this segment. If this segment contains the first or last point of the row, add the new label to the boundary array. This procedure is depicted below.

110220304 previously labeled data
110110101 raw data (second row)
110220304 labeled data (second row after labeling)
4. Count the number of points in each segment and add the number to the corresponding elements in the area array. The resultant arrays are shown below.

<table>
<thead>
<tr>
<th>label</th>
<th>number of points</th>
<th>boundary grain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

5. Using the last labeled row for comparison, extract the next row and repeat steps 3 and 4 until the last row of data is complete.

6. In the case where there are more than 1 label above a segment as shown below,

```
1 1 0 2 2 0 3 0 4  previously labeled row
1 1 0 2 2 0 3 0 4
0 0 0 0 0 0 1 1 1  raw data (third row)
0 0 0 0 0 0 3 3 3  labeled data
```

The segment is labeled with the first label encountered, 3 in this case, and merge the areas of the other labels to the first label encountered. The resulted arrays after this process becomes
7. Figure 18 (b) shows the labeled data file. The measured grain sizes are listed below.

<table>
<thead>
<tr>
<th>label</th>
<th>number of points</th>
<th>boundary grain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

8. Convert the area unit from points to conventional area units such as mm² or in².

Through this labeling and merging process, an area with any shape can be measured precisely. In the counting algorithm we introduced above, the fact that edges are part of the grain area is not considered. This problem can be easily solved by counting the number of the edge points between two segments and add half of the number to each segment. Another problem that should be taken into consideration
is the small false grains detected. These small grains can be easily found from the comparison of Figure 16(b), and Figure 17. This is a very common phenomenon if the edges in the picture are very thick (greater than 3 points). This problem can be reduced by neglecting the grains with size less than a preset value. The neglected size is distributed evenly to its adjacent grains.

Figure 19 shows the results of this counting process on the wireframe model shown in Figure 14(b) and the size measured from its original picture as shown in Figure 6 with a planimeter. The difference between the averages of these two measurements is shown to be less than 7%.

II-5 Least-Square Curve Fitting

The output data from the area counting process as shown in Figure 19 is a discrete curve. Using the least square criterion, we can fit a continuous function. By normalizing this function and taking the first order derivative, we can generate the probability density function of the distribution curve and its statistics. It is possible to obtain the probability density function by plotting the histogram of the measured grain sizes. However, the histogram is nothing but a flat line when plotted on a continuous area. If areas are grouped (i.e. all areas between 0.1 and 0.2 are considered to be the same sizes), the histogram conveys a larger spike representing the number of grains in this group. Since the area groups can be chosen randomly (i.e. 0.1 to 0.3 instead of 0.1 to 0.2), it is questionable how the data can be grouped to reflect the true density function.
Figure 19: Comparison between the grain sizes by planimeter and image analyzer.

- SIZE MEASURED WITH PLANIMETER
- SIZE MEASURED WITH THE ANALYZER

MEAN = .2558
S.D. = .1079

NUMBER OF GRAINS LESS THAN THE AREA

AREA (MM**2)
III Experimental Results

Figure 20 shows the system output of the sampled grain shown in Figure 17. The comparison of the system's average size output with that measured with a planimeter shows 3% in difference. This is much better than the 7% error as shown in section 2-4. The increase in accuracy can be anticipated when a larger number of grains are sampled. Figure 21 and 22 show the results of digitization, histogram segmentation, and edge detection processes on the samples taken from Figure 2. Figure 23 shows the combined results of the system output of the grains shown in Figure 4, 21a, and 23a. A polynomial of 7th order is fitted using the Least-Square criterion.

In some process such as melt crystallization, it may be desired to know not only the size distribution, but also the distribution changes with time. For these special purposes, the output data might be fitted into a particular type of distribution curve. By finding the essential parameters associated with this distribution, we see that the distribution changes with time can be expressed as parameter changes in time. Figure 25 shows an approximated Poisson distribution and Figure 26 shows the Poisson density function derived from Figure 25.

Table C-7 in Appendix C shows the grain sizes measured by the image analyzer without the operator's intervention, i.e., the region marks for the histogram segmentation process are selected directly from the program output without any modification. The discrepancy of the average size compared to that measured with a planimeter (0.2483 and 0.3165 mm², respectively) shows 22% in difference.
Figure 20 Comparison between the grain sizes measured by planimeter and image analyzer

- SIZE MEASURED WITH PLANIMETER
  - MEAN = 0.3165
  - S.D. = 0.3505
- SIZE MEASURED WITH THE IMAGE ANALYZER
  - MEAN = 0.3251
  - S.D. = 0.4315

Number of grains less than the area vs. area (mm^2)
Figure 21 Intermediate output from the image analyzer
(a) Grains sampled from Figure 2 (x390)
(b) Result of quantization with Chorus 6-bit digitizer
(c) Result of Histogram Segmentation
(d) Edges detected
Figure 22  Intermediate output from the image analyzer
(a) Grains sampled from Figure 2 (x390)
(b) Result of quantization with Chorus 6-bit digitizer
(c) Result of Histogram Segmentation
(d) Edges detected
ACCUMULATED GRAIN SIZE DISTRIBUTION CURVE

POLYNOMIAL CURVE FITTING

--- data obtained from the image analyzer

-- polynomial curve fitting

\[ f(x) = -63.61 + 1617.26x - 6882.78x^2 + \]
\[ 14959.80x^3 - 18839.24x^4 + \]
\[ 13347.86x^5 - 4961.99x^6 + \]
\[ 752.26x^7 \]

Figure 23  System output and polynomial curve fitting
GRAIN SIZE DISTRIBUTION DENSITY FUNCTION
POLYNOMIAL APPROXIMATION

\[ p(x) = 12.58 - 103.97x + 349.10x^2 - 586.18x^3 + 519.144x^4 - 231.59x^5 + 40.96x^6 \]

mean = 0.3471
S.D. = 0.4039

Figure 24  Probability density function of the sampled grains from Figure 2
ACCUMULATED GRAIN SIZE DISTRIBUTION CURVE

POISSON DISTRIBUTION CURVE FITTING

--- data obtained from the image analyzer

approximate Poisson distribution curve

\[ f(x) = \frac{ae^{-bx}}{b^2}(-bx - 1) \]

\[ a = 105.54 \]

\[ b = 10.27 \]

Figure 25 System output and Poisson distribution curve fitting
GRAIN SIZE DISTRIBUTION DENSITY FUNCTION
POISSON DISTRIBUTION APPROXIMATION

\[ p(x) = 105.54e^{-10.27x} \]

mean = 0.1947
S.D. = 0.1377

Figure 26 Poisson distribution density function derived from Figure 25
IV Concluding Remarks

In the previous sections, we discussed the general aspects of the computerized digital image analyzer. To complete the discussion, the following points are considered.

1. Is there a possibility for edge detection?

   Appendix A-5-1 shows that boundary edges must have high contrast for edge detection. This contrast can be improved by using the following techniques:
   a. Edge enhance techniques such as high-emphasis filtering, histogram equalization, and Laplacian operator discussed in Appendix A.
   b. Using microscopic aids such as filters and oblique illumination.
   c. Improvement of specimen preparation techniques such as etching and coloring.

2. Which parameters are to be evaluated?

   Parameters associated with a 2-D test plane such as mean intercept length, and size distribution can be obtained by this system. Three-dimensional parameters are not available in this system. Fortunately, many empirical equations have been derived to relate the measured 2-D parameter to the 3-D grain structure. These equations are available in most metallography text books.
3. What are the requirements in accuracy and speed?

At the present stage, the system is mainly software implemented. It requires a few minutes to complete one cycle of computation. Nevertheless, the output results are promising. The overall errors have been proven to be less than 10%. The error can be even less if a large number of samples are processed.

4. Can the specimen be moving?

Since the digitizer requires a finite segment of time for sampling and quantization, a fast moving specimen will result in a blurred digital image output from the digitizer. This problem can be solved by either using a high speed digitizer or a video recorder to capture the image and process afterwards.

5. Is the system required to be versatile?

Additional computations are usually required in some processes. This system includes a personal computer which is capable of many special applications such as speech control. The software implemented control program can be easily modified for any special applications.

Summarizing the above discussion, this system provides capabilities similar to that provided by an expensive analog image analyzer at considerably less expense. The personal computer in the system provides versatility. The lightweight equipment allows mobility. Best of all, since the personal computer and the micrograph are available in most metallurgical laboratories, the cost of adopting this system is approximately the cost of the digital data acquisition system. So far, this digital image analyzer is designed to be used
of the operator's expertise in determining the region marks for the histogram segmentation process. If a larger percentage of error, e.g. 20 percent, is allowed and the equipment used is upgraded, this digital image analyzer can be further improved to be fully automated.
BIBLIOGRAPHY


Appendix A

Introduction to Digital Image Processing

Image enhancement, as a required procedure for many picture processing algorithms, is a procedure for designing an operator which transforms the input picture to be of maximum utility for a specific application. There are three major problems encountered in a typical image enhancement problem, which are

1) noise
2) poor intensity distribution
3) degradation

As an example for demonstrating these problems, Figure A-1 shows a Westar VI satellite viewed from the space shuttle.

This section provides a fundamental introduction to some commonly used image enhancement techniques. Experiments were conducted under the contract NAS9-17145 from the TV section of the Tracking and Communications Division of the NASA Johnson Space Center.

A-1 Sampling and Quantization

Sampling, in general, can be represented as variable or fixed rate sampling as depicted in Figure A-2.

If a picture \( f(x,y) \) has cutoff frequencies of \( \omega_x \) and \( \omega_y \), then the sampled image can be expressed as

\[
f_s(i\Delta x, j\Delta y) = \sum_{i=1}^{M} \sum_{j=1}^{N} f(x,y) \delta(x - i\Delta x, y - j\Delta y) \quad (A.1a)
\]

for a fixed sampling rate, or
Figure A-1  Westar VI satellite viewed from the space shuttle

Figure A-2  Fixed and variable sampling schemes
Figure A-3 Example of 4-bit quantization
Array size: 320 x 200

Figure A-4 Example of 6-bit quantization
Array size: 640 x 400
Figure A-5 A typical image formation system

Figure A-6 Image with additive white noise ($\rho^2 = 200$)
\[ f_s(i\Delta x, j\Delta y) = \sum_{i=1}^{M} \sum_{j=1}^{N} f(x,y) \delta(x - 3(i\Delta x), y - b(j\Delta y)) \quad (A.1b) \]

for a variable sampling rate, where \( \Delta x < \frac{1}{2} \omega_x \) and \( \Delta y < \frac{1}{2} \omega_y \).

The output digital picture \( f_s(i,j) \) is a rectangular array of integers. Each element of a digital picture is called a pixel, and the value of a pixel is called graylevel. The graylevel of a pixel represents the average light intensity of the portion of the picture represented by the pixel. Graylevel is an \( N \)-bit number depending on the quantization machine which also determines the resolution of the digital picture. A 6-bit digitizer produces a digital picture of 64 shades or levels of gray. Figure A-3 and A-4 show the results of fixed rate sampling with 4-bit and 6-bit quantization, respectively.

A-2 Noise Measurement

For most applications, it is desired to be able to evaluate the noise parameters associated with the image data. A typical image formation scheme is depicted in Figure A-5.

If we assume the noise is uncorrelated with the picture, we then model the noisy picture \( g(x,y) \) and \( g'(x,y) \) as

\[
g(x,y) = f(x,y) * h(x,y) + n(x,y) \quad (A-2a)\]

\[
g'(x,y) = (f(x,y) * h(x,y)) \ h(x,y) \quad (A-2b)\]

where \( (A * B) \) means convolution of \( A \) with \( B \).

Thus the first measurement of the noise is to determine whether it is an additive or multiplicative type of noise, and the second measurement of the noise is its density distribution as well as its amplitude.
Equation A-2b shows that if the real world scene is monotonous, i.e. \( f(x,y) = 0 \), then the observed picture \( g'(x,y) \) will not be affected by the noise, however, this is not true in real situations. Thus it is reasonable to assume the presence of the additive type of noise in the observed noisy picture.

Figure A-6 shows an image with additive white noise.

It is not a trivial problem to determine the exact characteristics of the additive noise. However, it is easy to show that the spectrum of a random noise has higher response in the high frequency region. Figure A-7 shows a 1-D sequence of random noise on a flat background and its spectrum. This general characteristic of random noise suggests that random noise can be removed by means of low pass filtering or spatial averaging.

A-3 Edge Enhancement

As indicated in Figure A-1, the edge information contained in the picture is not obvious to the human eye. Therefore, it is necessary to readjust the light intensity distribution. Several techniques which are designed for this purpose are discussed in the following sections.

A-3-1 Histogram Equalization [3]

HE (histogram equalization) is a spatial domain processing technique which rescales the graylevels of each pixel with the information obtained from the histogram of the picture. Consider the HE operator as a function
Figure A-7  Spectrum of a sequence of random noise

Figure A-8  Normalized histogram

Figure A-9  Mapping function calculated from Figure A-8
\[ Z = f(v) \quad (A-3.1a) \]

where \( Z \) = output graylevel

\( v \) = input graylevel

Thus the problem is to choose a single-valued mapping function \( f(v) \) which has the limits of

\[ Z_{\text{min}} \geq 0, \text{ and} \]

\[ Z_{\text{max}} < \infty \]

To introduce the HE procedure, let us assume that the histogram of the intensity function \( p(v) \) in which \( v \) is normalized from 0 to 1 as shown in Figure A-8 is a continuous function of \( v \).

The integral of \( p(v) \) with respect to \( v \) from 0 to 1 is depicted in Figure A-9. If one writes

\[ Z = f(v) = \int_{0}^{v} p(v) \, dv \quad (A-3.1b) \]

Then \( f(v) \) is the desired correction function which has the limits of

\[ 0 \leq f(v) \leq 1 \quad \text{for } 0 \leq v \leq 1 \]

For a discrete density function, \( f(v) \) can be expressed as

\[ f(v) = \Sigma p(v) \quad (A-3.1c) \]
Figure A-10 Result of Histogram Equalization on Figure A-1

Figure A-11 Histogram of Figure A-1
Figure A-12 Histogram of Figure A-10
Figure A-10 shows the result of the application of this technique to the picture of Figure A-1. Figure A-11 and Figure A-12 show the histograms of the original picture and the equalized picture.

A-3-2 Laplacian Operator

The Laplacian operator is defined [8] as

$$f(x,y) = s(x,y) - \nabla^2 s(x,y)$$  \hspace{1cm} (A-3.2a)

Assume the edge information in a picture is blurred as the result of a diffusion process which is governed by the well known differential equation of population growth

$$\frac{\partial g}{\partial t} = k \nabla^2 s$$  \hspace{1cm} (A-3.2b)

where \( g = g(x,y,t) \) and \( f(x,y) = g(x,y,0) \) is the unblurred picture.

Expanding \( g(x,y,t + \tau) \) by using Taylor series expansion and let \( t = 0 \)

$$g(x,y,0) = g(x,y,\tau) - \tau \frac{\partial g(x,y,t)}{\partial t} - \tau^2 \frac{\partial^2 g(x,y,t)}{\partial^2 t} - \ldots$$

Assume \( \tau \) is small. Then the quadratic and higher order terms can be neglected. Substituting the results in eq. A-3.2b, we get the definition of the Laplacian operator. In the discrete case, let us define the second derivative as
Figure A-13  Geometrical model of a space satellite

Figure A-14  Result of the Laplacian operation on Figure A-13
\[ D^2\{g(m,n)\} = g(m+1,n) + g(m-1,n) + g(m,n+1) + g(m,n-1) - 4g(m,n) \]

Eq. A-3.2a can be expressed as

\[ f(m,n) = g(m,n) + 5g(m,n) - \frac{1}{5} \left[ \sum_{i=1}^{M} \sum_{j=1}^{N} g(m+i,n+j) \right] \]  \hspace{1cm} (A-3.2c)

Examination of eq. A-3.2c shows that the Laplacian operator is a spatial domain high-emphasis filter. This is recognized by the addition of the observed picture \( g(m,n) \) to its high frequency components. Figure A-14 shows the result of this operation on Figure A-13.

A-3-3  **High Emphasis Filtering**

HEF (high emphasis filtering) is similar to the Laplacian operator but it is a process in the spatial frequency domain. To design a high emphasis filter, one can simply convert a low pass filter, which will be discussed later, using the equation

\[ F(u,v) = K\{G(u,v)\} - L\{G(u,v)\} \]  \hspace{1cm} (A-3.3a)

where \( K = \text{gain factor constant} > 1 \)

\( L\{ \} = \text{low pass filtering operator} \)

or use the difference of two Gaussian functions (DOG),

\[ H(u,v) = A \exp\left(- \frac{||u,v||^2}{2 \sigma_1^2}\right) - B \exp\left(- \frac{||u,v||^2}{2 \sigma_2^2}\right) \]  \hspace{1cm} (A-3.3b)

where
\[ H(u,v) = \text{psf (point spread function)} \] of the HEF operator

\[ A, B \quad \text{gain factor constants and } A \Rightarrow B \]

\[ \sigma_1, \sigma_2 \quad \text{variance constants and } \sigma_1 \leq \sigma_2 \]

To see the difference between the two approaches, let's examine the psf's of a converted Butterworth low pass filter, as shown in Figure A-15, and the DOG operator, as shown in Figure A-16, in frequency domain.

Figure A-15 shows a purely HEF but Figure A-16 indicates the DOG results in a high-emphasis-low-pass filter.

A further study of the behavior of the DOG operator indicates

1) by increasing the difference between \( \sigma_1 \) and \( \sigma_2 \), the slope of the edge function increases but the "overshoot" effect is decreased.

2) by increasing the difference between \( A \) and \( B \), both the slope and the "overshoot" effect increase.

A-3-4 Homomorphic Filtering [3]

Homomorphic filtering is designed for increasing the intensity contrast of pictures. To illustrate Homomorphic filtering, let us assume the observed picture is composed of

\[ f(x,y) = r(x,y) i(x,y) \quad \text{(A-3.4a)} \]

where \( r(x,y) = \text{reflectance function} \)

\( i(x,y) = \text{illumination function} \)

By taking a logarithmic operation, eq. A-3.4a becomes

\[ \log(f(x,y)) = \log(r(x,y)) + \log(i(x,y)) \quad \text{(A-3.4b)} \]
Figure A-15 High emphasis filter derived from Eq. A-3.3a
Figure A-16 High emphasis filter derived from Eq. A-3.3b

Figure A-17 Result of Homomorphic filtering with a Butterworth high emphasis filter
If \( i(x,y) \) is considered approximately constant, which is reasonable if the object is small compared with its distance from the illumination sources. Thus by taking the Fourier Transform of eq. A-3.4b, one can expect that \( \log(i(x,y)) \) contributes mainly in the low frequency region and \( \log(r(x,y)) \) contributes mainly in the high frequency region in the frequency domain. Homomorphic filtering is a process which tries to decrease the illumination while increasing the reflectance of the picture by using a high emphasis filter with gain factor less than 1. This operation indeed is trying to make the edges more visible at the expense of losing the overall brightness of the picture. Figure A-17 shows the results of this operation on Figure A-1.

A-4 Noise Removal (Restoration)

In section A-2, noise contained in a noisy picture is assumed to be additive random noise and the noisy picture is modeled as

\[
g(x,y) = H\{f(x,y)\} + n(x,y) \tag{A-4}\]

where \( H \{ \} \) = the degradation process.

In this section, several techniques that are designed for diminishing the noise are discussed.

A-4-1 Unweighted Windowing (Averaging)

Averaging is a spatial domain degradation process which convolves the picture function \( f(x,y) \) with a square wave function
\[ S(x, y) = \begin{cases} 1/T^2 & |x| \to T/2, \quad |y| \to T/2 \\ 0 & \text{else} \end{cases} \]

To see how this operation behaves, let us look at the psf of this function, that is

\[ H(u, v) = \frac{\sin \pi u T}{\pi u T} \frac{\sin \pi v T}{\pi v T} \tag{A-4.1} \]

Figure A-18 shows eq. A-4.1 graphically and shows that averaging indeed is a low pass filtering operation, which is an integration process in the spatial domain. Therefore, we can expect that this operation will smear the noise at the expense of losing the sharpness of the processed picture.

Figure A-19 and A-20 shows the results of this processing.

A-4-2 **Low Pass Filtering**

Low pass filtering has the same properties as the averaging technique but is a process in the frequency domain. There are three most commonly used low pass filters, which are

1) Trapezoid LPF
2) Butterworth LPF
3) Exponential LPF

In order to evaluate the behavior of a low pass filter (LPF), let us consider an ideal 1-D LPF as depicted in Figure A-21.
Figure A-18  Square windowing function in spatial frequency domain
Figure A-19 Result of 3x3 windowing on Figure A-10

Figure A-20 Result of 5x5 windowing on Fig. A-10
Note that the ideal LPF is a sinc function in spatial domain which results in "echoes" in the picture after processing. To prevent this undesired effect, a practical LPF should be a piecewise smooth function such as Butterworth and Exponential LPF, which are given as

\[ h(u,v) = \frac{1}{1 + \left[ \frac{r(u,v)}{r_0} \right]^{2n}} \quad (A-4.2a) \]

and

\[ h(u,v) = \exp(-r/r_0)^n \quad (A-4.2b) \]

respectively, where \( r_0 \) is the cutoff frequency.

\[ r = \sqrt{u^2 + v^2} \]

To give a close examination of the difference between averaging and low pass filtering, one can take the inverse Fourier Transform of eq. A-4.2a and compare the results with the window used in averaging as shown in Figure A-21.

As shown in Figure A-21, an LPF can be expressed as a weighted averaging process.

A-4-3 Wiener Filtering (Least Squares Filtering)

Consider the degraded noisy picture model introduced in section A-2, that is

\[ g(x,y) = H[f(x,y)] + n(x,y) \quad (A-4.3a) \]
Figure A-21 Ideal low pass filter

Figure A-22 Result of Butterworth LPF with
n = 1
r = .7 total energy
Figure A-23 Result of Butterworth LPF with
n = 1
r = .5 total energy
Let \( \hat{f}(x, y) \) denote the estimate for the undegradated picture function \( f(x, y) \) and define the error of the estimate as

\[
e = f(x, y) - \hat{f}(x, y)
\]

(A-4.3b)

If \( \hat{f}(x, y) \) is obtained through a linear position invariant operation

\[
\hat{f}(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} w(x - d, y - \beta)g(\alpha, \beta)d\alpha d\beta
\]

(A-4.3c)

then the goal is to find an operator \( w(x, y) \) such that equation A-4.3b is minimized.

By employing the Minimum Mean Square Error criterion, that is

\[
e = E\{ (f(x, y) - \hat{f}(x, y))^2 \}
\]

(A-4.3d)

or

\[
e = E\{ (f(x, y) - \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} w(x - \alpha, y - \beta)s(d, \beta)d\alpha d\beta) \}
\]

(A-4.3e)

A. Rosenfield and A. Kak [8] showed that a function \( w(x, y) \) satisfying

\[
E\{ (f(x, y) - \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} w(x - \alpha, y - \beta)g(\alpha, \beta)d\alpha d\beta)g(x', y') \} = 0
\]

(A-4.3f)

will minimize eq. A-4.3e. They prove that

\[
W(u, v) = \frac{H^*(u, v)}{|H(u, v)|^2 + \frac{S_{nn}(u, v)}{S_{ff}(u, v)}}
\]

(A-4.3g)

In the derivation of eq. A-4.3g, it is assumed that
1) Noise is determined to be additive Gaussian white noise with zero mean.

2) The power spectral density functions \( S(u,v) \) of noise and picture functions are known.

3) The degraded process \( h(x,y) \) is also known.

In the cases where this \textit{a priori} knowledge is unknown, we must conduct a procedure for estimating these parameters. In our experiments, we separated the degraded picture into two subpictures according to its local spatial activities (gradient of intensity change). Let us define the first order derivative in the discrete case as

\[
D[f(x,y)] = \sqrt{\text{sqr}[f(x+1,y)+f(x+2,y)-f(x-1,y)-f(x-2,y)]}
+ \text{sqr}[f(x,y+2)+f(x,y+1)-f(x,y-1)-f(x,y-2)]}
\]  \hspace{1cm} (A-4.3h)

This definition was adopted for depressing the effect of noise. Since each subpicture contains different information, we make the following assumptions accordingly:

1) Subpicture 1 includes those pixels in the lower gradient region. This implies that it contains the information for flat surfaces and slowly varying surface boundaries. We then assume this region is noise free, \( S_{nn}(u,v) = 0 \), and degraded with a low pass filter.

2) Subpicture 2 includes those pixels within the higher gradient region. This implies that it contains the information of sharp edges and the random noise. We then
assume this region is undegradated, \( H(u,v) = 1 \), and the
noise-signal ratio \( S_{nn}/S_{ff} \) equals to a constant \( 2\sigma^2 \),
where \( \sigma \) is the variance of this subpicture.

Based on the above assumptions, we process each subpicture using eq.
A-4.3g and combine the results.

Figure A-25 shows the result of this process on Figure A-24.

Alternatively, if a portion of the noisy picture \( \Omega \) can be
extracted in which only noise is present, the power spectrum of the
noise \( S_{nn} \) can be estimated from it. The power spectrum of the
undegraded picture \( S_{ff} \) is assumed equal to \( S_{gg} \), the power spectrum
of the noisy picture. An iteration process using equation A-4.3g with
these estimated noise-signal ratio can be developed for noise
removal. After each iteration, the difference between \( S_{ff} \) and \( S_{gg} \) is
converging while the noise spectrum \( S_{nn} \) is getting smaller.

A-5 Edge Detection (Segmentation)

In this section, we discuss some segmentation methods which are
designed for identifying region boundaries (edges) of surfaces in the
observed picture. We define the edge as the boundary of two regions
with different graylevels (light intensities). We say that an edge is
present is the difference of graylevels exceeds a preset threshold
value. The following techniques for detecting edges were conducted in
our experiments.
Figure A-24 Geometrical model of satellite with random noise

Figure A-25 Result of Wiener filtering on Figure A-24
Spatial domain differentiation with thresholding

In the spatial domain, edge detectors may be categorized into two areas:

1) First derivative process as shown in Figure A-26(b). Edges are obtained by searching for the local maximum and depressing the others. Alternatively, edges can be obtained by comparing the gradient with a preset threshold value \( T \) and selecting those points at which the gradients are larger than \( T \).

2) Second derivative process as shown in Fig. A-26(c). Edges are obtained by searching for those zero-crossing points.

Since both kinds of operators use the local edge information, the results are usually dominated by the presence of noise. This is the reason for using a preprocess for noise removal as discussed in the previous section.

In our experiments, a Sobel edge detector was used for edge detection. It is a first derivative process. The gradient at a point is defined as the weighted difference of its two adjacent regions. Fig. A-27 shows the weights of a region of \( 9 \) (3 x 3) pixels.

By convolving these two operators with the processed pictures and summing the results, we obtained a corresponding picture in the gradient domain. Edges were obtained by comparing the gradients with a preset threshold value. Figure A-28 shows the result of this process.
Figure A-26 (a) Edge model (b) First derivative (c) Second derivative

\[
\begin{bmatrix}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1
\end{bmatrix}
\]

Figure A-27 Sobel operators
Figure A-28  
(1) Geometrical Model  
(2) Result of Wiener filtering  
(3) Result of sobel edge detector  
(4) Result of sobel edge detector with smaller threshold value T

Figure A-29  Geometrical representation of eq. A-5.2a
Template matching is a feature matching process in the spatial domain. It decomposes the processed picture into a summation of subpictures in which only one feature (edge, line, point, etc) is presented in each subpicture. Through this representation, it is required to determine what type of feature is contained in a finite subregion of the processed picture. Let us consider a small subregion of size 3x3 for the processed picture and represent it in lexicographic form, that is
\[
g(x,y) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}
\]
then,
\[
s(\vec{r}) = \begin{bmatrix} r_1^T \\ r_2^T \\ r_3^T \end{bmatrix}
\]
where \( \vec{r}_n = r_{n1} \, r_{n2} \, r_{n3} \)

We need to find these coefficients \( x_i \) so that
\[
g(\vec{r}) = \sum_{i=1}^{N} x_i \phi_i(\vec{r}) \quad (A-5.2a)
\]
where \( \phi_i(\vec{r}) \) is the orthogonal basis represent different features.

Let \( N=3 \), eq. A-5.2a is expressed in geometrical representation as depicted in Figure A-29.

From Figure A-29 we see that these coefficients \( x_n \) can be computed through the inner products of \( s(\vec{r}) \) with \( \phi_n(\vec{r}) \), that is,
\[ x_n = s(\hat{r}) \phi_n(\hat{r}) \quad (A-5.2b) \]

We then determine the type of feature presenting in the subregion by choosing the feature associated with the largest coefficient. We can also set up threshold values for each basis and say a feature \( n \) presents if \( x_n \) exceeds its threshold value.

In our experiments, we chose a set of nine orthogonal templates as shown in Figure A-30. We formed an edge subspace in which

\[ x_e = \text{SQRT} \left( \sum_{i=1}^{4} x_i^2 \right) \quad (A-5.2c) \]

We formed a line subspace in which

\[ x_1 = \text{SQRT} \left( \sum_{i=5}^{8} x_i^2 \right) \quad (A-5.2d) \]

and third subspace \( x_9 \) represents the feature of flat region. Figure A-32 and A-33 show the results of this process on a test pattern shown in Figure A-31.

A-6 Summary

Figure A-34 shows a flow chart of a typical image processing unit for pattern recognition. It is introduced as a summary of the image processing techniques discussed in the previous sections. Appendix D contains the software implementation of the experiments conducted.
Figure A-30 Orthogonal templates
Figure A-31 Test pattern for edge detector
Figure A-32  Projection of the test pattern on edge subspace

Figure A-33  Projection of the test pattern on line subspace
Figure A-34 Flow chart of the image processing algorithms
Appendix B

Introduction to Chorus "PC1000" 6-Bit Digitizer
and IBM-PC Memory Management

The software driven Chorus 6-bit digitizer is a fixed rate sampling and quantization mechanism. It converts an analog picture transmitted from the TV camera into a digital picture. The digital picture is a rectangular array of integers. Each element of a digital picture is called a pixel. The value of a pixel is called the graylevel. The graylevel of a pixel represents the average light intensity of the portion of the picture (grid) represented by that pixel. Physically, the graylevel of a pixel is obtained by:

(a) Measuring the voltage of a grid output from the TV camera.
(b) Comparing this voltage with the preset quantization scale, i.e. the upper and lower bound.
(c) If this voltage exceeds the upper bound, the pixel associated with this grid is considered as "white". If this voltage is below the lower bound, it is considered as "black".
(d) Voltage between the bounds represents the range of graylevels.

The Chorus digitizer quantizes each pixel to one of 64 graylevels, i.e. a 6-bit number. "Black" and "white" are represented by 0 and 63, respectively. Numbers between 0 and 63 represent different levels of gray. The output digital picture from the Chorus digitizer can be displayed on the CRT and stored in memory or on a
disk. A byte of memory is used to store the graylevel of a pixel. 250k-bytes of memory are required to store a 400 x 640 digital picture. The digital picture is stored row by row sequentially into memory starting at a known address. Equations B-1 and B-2 calculate the address containing the graylevel of a particular pixel in the digital picture as shown in Figure B1.

\[
\text{offset} = (\text{number of pixels per row}) \times i + j \quad \text{(B-1)}
\]
\[
\text{address} = \text{starting address} + \text{offset} \quad \text{(B-2)}
\]

For example, if a 400 x 640 digital picture is stored in memory starting at address 30000 in hexadecimal, and we want to know what is the address of the last pixel, \(i = 399\) and \(j = 639\). From equations B-1 and B-2

\[
\text{offset} = 640 \times 399 + 639 = 255999 \quad \text{in decimal}
\]
\[
= 3FFFF \quad \text{in hexadecimal}
\]
\[
\text{address} = 30000 + 3FFFF = 6FFFF \quad \text{in hexadecimal}
\]

The IBM-PC has an 8088 chip as the CPU. This chip allows 16-bit address accessibility. This means that the largest address number that can be accessed is 65535 or FFFF in hexadecimal. In order to overcome this limitation, memory larger than 64k-bytes is subdivided into segments. A segment is a contiguous area of memory up to 64k-bytes in length. Addresses larger than 65535 are accessed by finding the segment in which the address is contained and the offset from the beginning of the segment. For this reason, addresses larger than 65535 are expressed in the form of segment:offset. Equations B-3 and B-4 compute the segment and offset for a known address.
offset = address mod 65536 \hspace{1cm} (\text{B-3})

segment = address-offset/16 \hspace{1cm} (\text{B-4})

where mod means modulation, i.e. A mod B = A - integer (A/B).

Continuing the previous example, the segment and offset of the address 6FFFF or 458751 in decimal are found as following:

\[
\begin{align*}
\text{offset} &= 458751 \mod 65536 = 65535 \text{ in decimal} \\
&= \text{FFFF in hexadecimal} \\
\text{segment} &= (458751-65535)/16 = 24576 \text{ in decimal} \\
&= \text{6000 in hexadecimal}
\end{align*}
\]

This address is expressed as 6000:FFFF. This expression can be interpreted as that the address 6FFFF is contained in the segment 6000 and offset FFFF from the beginning of this segment. The beginning address of each segment is calculated internally by multiplying 16 with the segment identifier.
Appendix C
System Output Data

This section includes the experimental results of the grain sizes measured by using a planimeter and the system output data for the grain structures shown in Figures 14, 16, 21 and 22. Table C-7 shows the grain sizes measured by the system without the operator's intervention. Table C-8 shows a portion of the wireframe picture data file. Figure C-1 and C-2 depict the smoothed histograms and the local minimums selected by the program.
Table C-1 Grain size measurements output for the grain structure model shown in Figure 14

<table>
<thead>
<tr>
<th>grain identifier</th>
<th>area (mm$^2$) measured by planimeter</th>
<th>area (mm$^2$) direct counting on data file</th>
<th>system output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1915</td>
<td>0.1887</td>
<td>0.2043</td>
</tr>
<tr>
<td>2</td>
<td>0.2873</td>
<td>0.2898</td>
<td>0.3035</td>
</tr>
<tr>
<td>3</td>
<td>0.4052</td>
<td>0.4062</td>
<td>0.4283</td>
</tr>
<tr>
<td>4</td>
<td>0.0976</td>
<td>0.1215</td>
<td>0.1160</td>
</tr>
<tr>
<td>5</td>
<td>0.2652</td>
<td>0.2660</td>
<td>0.2851</td>
</tr>
<tr>
<td>6</td>
<td>0.4954</td>
<td>0.4730</td>
<td>0.5093</td>
</tr>
<tr>
<td>7</td>
<td>0.3113</td>
<td>0.3504</td>
<td>0.3472</td>
</tr>
<tr>
<td>8</td>
<td>0.3094</td>
<td>0.2953</td>
<td>0.3320</td>
</tr>
<tr>
<td>9</td>
<td>0.1819</td>
<td>0.1773</td>
<td>0.1914</td>
</tr>
<tr>
<td>10</td>
<td>0.1529</td>
<td>0.1753</td>
<td>0.1746</td>
</tr>
<tr>
<td>11</td>
<td>0.0215</td>
<td>0.0402</td>
<td>0.0391</td>
</tr>
<tr>
<td>12</td>
<td>0.1446</td>
<td>0.1621</td>
<td>0.1613</td>
</tr>
<tr>
<td>13</td>
<td>0.2007</td>
<td>0.1961</td>
<td>0.2179</td>
</tr>
<tr>
<td>14</td>
<td>0.2449</td>
<td>0.2324</td>
<td>0.2449</td>
</tr>
<tr>
<td>15</td>
<td>0.2684</td>
<td>0.2680</td>
<td>0.2871</td>
</tr>
<tr>
<td><strong>MEAN</strong></td>
<td><strong>0.2389</strong></td>
<td><strong>0.2428</strong></td>
<td><strong>0.2558</strong></td>
</tr>
<tr>
<td><strong>STANDARD DEVIATION</strong></td>
<td><strong>0.1189</strong></td>
<td><strong>0.1114</strong></td>
<td><strong>0.1079</strong></td>
</tr>
<tr>
<td>Area</td>
<td>Grains</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.400000 mm²</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.019531 mm² (50 pixrs)</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.410034</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3837891 mm²</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.599670</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Total number of grains:** 40

**Boundary grains:** 25

**Inner grains:** 15

**The grain size distribution:**

<table>
<thead>
<tr>
<th>Area (mm²)</th>
<th>Grains</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.906E-2</td>
<td>1</td>
</tr>
<tr>
<td>4.160E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.613E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.746E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.914E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.043E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.179E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.449E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.651E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.871E-1</td>
<td>1</td>
</tr>
<tr>
<td>3.035E-1</td>
<td>1</td>
</tr>
<tr>
<td>3.320E-1</td>
<td>1</td>
</tr>
<tr>
<td>3.472E-1</td>
<td>1</td>
</tr>
<tr>
<td>4.238E-1</td>
<td>1</td>
</tr>
<tr>
<td>5.093E-1</td>
<td>1</td>
</tr>
</tbody>
</table>

**The average area per grain:**

2.558E-1 mm²

**The standard deviation of the distribution curve:**

1.077E-1 mm²

**The average grains per mm²:**

3.908397 grains
Table C-3 Grain size measurements output for the grain structure shown in Figure 16

<table>
<thead>
<tr>
<th>grain identifier</th>
<th>area (mm**2)</th>
<th>measured by planimeter</th>
<th>system output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*</td>
<td></td>
<td>0.0582</td>
</tr>
<tr>
<td>2</td>
<td>*</td>
<td></td>
<td>0.0590</td>
</tr>
<tr>
<td>3</td>
<td>*</td>
<td></td>
<td>0.0594</td>
</tr>
<tr>
<td>4</td>
<td>*</td>
<td></td>
<td>0.0617</td>
</tr>
<tr>
<td>5</td>
<td>*</td>
<td></td>
<td>0.0641</td>
</tr>
<tr>
<td>6</td>
<td>*</td>
<td></td>
<td>0.0652</td>
</tr>
<tr>
<td>7</td>
<td>*</td>
<td></td>
<td>0.0680</td>
</tr>
<tr>
<td>8</td>
<td>0.0160</td>
<td></td>
<td>0.0719</td>
</tr>
<tr>
<td>9</td>
<td>0.0353</td>
<td></td>
<td>0.0754</td>
</tr>
<tr>
<td>10</td>
<td>0.0417</td>
<td></td>
<td>0.0781</td>
</tr>
<tr>
<td>11</td>
<td>0.0449</td>
<td></td>
<td>0.0789</td>
</tr>
<tr>
<td>12</td>
<td>0.0738</td>
<td></td>
<td>0.0809</td>
</tr>
<tr>
<td>13</td>
<td>0.0867</td>
<td></td>
<td>0.0895</td>
</tr>
<tr>
<td>14</td>
<td>0.0899</td>
<td></td>
<td>0.0930</td>
</tr>
<tr>
<td>15</td>
<td>0.0931</td>
<td></td>
<td>0.1015</td>
</tr>
<tr>
<td>16</td>
<td>0.0963</td>
<td></td>
<td>0.1027</td>
</tr>
<tr>
<td>17</td>
<td>0.0963</td>
<td></td>
<td>0.1082</td>
</tr>
<tr>
<td>18</td>
<td>0.1027</td>
<td></td>
<td>0.1156</td>
</tr>
<tr>
<td>19</td>
<td>0.1156</td>
<td></td>
<td>0.1160</td>
</tr>
<tr>
<td>20</td>
<td>0.1509</td>
<td></td>
<td>0.1285</td>
</tr>
<tr>
<td>21</td>
<td>0.1605</td>
<td></td>
<td>0.1339</td>
</tr>
<tr>
<td>22</td>
<td>0.1605</td>
<td></td>
<td>0.1359</td>
</tr>
<tr>
<td>23</td>
<td>0.1733</td>
<td></td>
<td>0.1609</td>
</tr>
<tr>
<td>24</td>
<td>0.1765</td>
<td></td>
<td>0.1621</td>
</tr>
<tr>
<td>25</td>
<td>0.1990</td>
<td></td>
<td>0.1703</td>
</tr>
</tbody>
</table>

* grains that can not be found in the original picture
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>0.2006</td>
<td>0.1710</td>
</tr>
<tr>
<td>27</td>
<td>0.2183</td>
<td>0.1859</td>
</tr>
<tr>
<td>28</td>
<td>0.2247</td>
<td>0.1960</td>
</tr>
<tr>
<td>29</td>
<td>0.2247</td>
<td>0.2046</td>
</tr>
<tr>
<td>30</td>
<td>0.2439</td>
<td>0.2179</td>
</tr>
<tr>
<td>31</td>
<td>0.2632</td>
<td>0.2210</td>
</tr>
<tr>
<td>32</td>
<td>0.2696</td>
<td>0.2429</td>
</tr>
<tr>
<td>33</td>
<td>0.2760</td>
<td>0.2761</td>
</tr>
<tr>
<td>34</td>
<td>0.2985</td>
<td>0.2835</td>
</tr>
<tr>
<td>35</td>
<td>0.3017</td>
<td>0.3011</td>
</tr>
<tr>
<td>36</td>
<td>0.3370</td>
<td>0.3113</td>
</tr>
<tr>
<td>37</td>
<td>0.3499</td>
<td>0.3386</td>
</tr>
<tr>
<td>38</td>
<td>0.3659</td>
<td>0.3738</td>
</tr>
<tr>
<td>39</td>
<td>0.3691</td>
<td>0.4746</td>
</tr>
<tr>
<td>40</td>
<td>0.3691</td>
<td>0.6097</td>
</tr>
<tr>
<td>41</td>
<td>0.5071</td>
<td>0.6531</td>
</tr>
<tr>
<td>42</td>
<td>0.5296</td>
<td>0.6710</td>
</tr>
<tr>
<td>43</td>
<td>0.6580</td>
<td>0.6730</td>
</tr>
<tr>
<td>44</td>
<td>0.7511</td>
<td>0.7394</td>
</tr>
<tr>
<td>45</td>
<td>0.9886</td>
<td>0.8566</td>
</tr>
<tr>
<td>46</td>
<td>1.1523</td>
<td>1.1380</td>
</tr>
<tr>
<td>47</td>
<td>1.8487</td>
<td>1.6770</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MEAN</td>
<td>0.3165</td>
</tr>
<tr>
<td>STD</td>
<td>0.3505</td>
</tr>
<tr>
<td>DEV</td>
<td></td>
</tr>
</tbody>
</table>
### Table C-4 System output data for grain structure shown in Figure 16

<table>
<thead>
<tr>
<th>The area of processed picture:</th>
<th>20.000000 mm² * 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>The grains of size less than</td>
<td>0.039663 mm² * 2 (100 pixels) are ignored but the loss is compensated into each output grains</td>
</tr>
<tr>
<td>Total number of grains :</td>
<td>92</td>
</tr>
<tr>
<td>Boundary grains :</td>
<td>44</td>
</tr>
<tr>
<td>2.410547 mm² * 2 (0.120327)</td>
<td></td>
</tr>
<tr>
<td>Inner grains :</td>
<td>48</td>
</tr>
<tr>
<td>15.608984 mm² * 2 (0.780449)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The grain size distribution :</th>
</tr>
</thead>
<tbody>
<tr>
<td>area (mm²)</td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>5.820E-2</td>
</tr>
<tr>
<td>5.898E-2</td>
</tr>
<tr>
<td>5.937E-2</td>
</tr>
<tr>
<td>6.171E-2</td>
</tr>
<tr>
<td>6.406E-2</td>
</tr>
<tr>
<td>6.523E-2</td>
</tr>
<tr>
<td>6.796E-2</td>
</tr>
<tr>
<td>7.187E-2</td>
</tr>
<tr>
<td>7.539E-2</td>
</tr>
<tr>
<td>7.812E-2</td>
</tr>
<tr>
<td>8.590E-2</td>
</tr>
<tr>
<td>8.958E-2</td>
</tr>
<tr>
<td>9.296E-2</td>
</tr>
<tr>
<td>1.015E-1</td>
</tr>
<tr>
<td>1.027E-1</td>
</tr>
<tr>
<td>1.082E-1</td>
</tr>
<tr>
<td>1.156E-1</td>
</tr>
<tr>
<td>1.160E-1</td>
</tr>
<tr>
<td>1.285E-1</td>
</tr>
<tr>
<td>1.337E-1</td>
</tr>
<tr>
<td>1.359E-1</td>
</tr>
<tr>
<td>Value</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>1.609E-1</td>
</tr>
<tr>
<td>1.621E-1</td>
</tr>
<tr>
<td>1.703E-1</td>
</tr>
<tr>
<td>1.710E-1</td>
</tr>
<tr>
<td>1.859E-1</td>
</tr>
<tr>
<td>1.960E-1</td>
</tr>
<tr>
<td>2.046E-1</td>
</tr>
<tr>
<td>2.179E-1</td>
</tr>
<tr>
<td>2.210E-1</td>
</tr>
<tr>
<td>2.429E-1</td>
</tr>
<tr>
<td>2.761E-1</td>
</tr>
<tr>
<td>2.835E-1</td>
</tr>
<tr>
<td>3.011E-1</td>
</tr>
<tr>
<td>3.113E-1</td>
</tr>
<tr>
<td>3.386E-1</td>
</tr>
<tr>
<td>3.738E-1</td>
</tr>
<tr>
<td>4.746E-1</td>
</tr>
<tr>
<td>6.097E-1</td>
</tr>
<tr>
<td>6.531E-1</td>
</tr>
<tr>
<td>6.710E-1</td>
</tr>
<tr>
<td>6.730E-1</td>
</tr>
<tr>
<td>7.394E-1</td>
</tr>
<tr>
<td>8.566E-1</td>
</tr>
<tr>
<td>1.138E0</td>
</tr>
<tr>
<td>1.677E0</td>
</tr>
</tbody>
</table>

The average area per grain:
3.251E-1 mm²

The standard deviation of the distribution curve:
4.315E-1 mm²

The average grains per mm²:
3.075152 grains
Table C-5 System output data for grain structure shown in Figure 21

The area of processed picture:
20.000000 mm**2

The grains of size less than 0.039063 mm**2 (100 pixels) are ignored but the loss is compensated into each output grains

Total number of grains : 91

Boundary grains : 40
4.496094 mm**2 (0.224805)

Inner grains : 51
14.201172 mm**2 (0.710059)

The grain size distribution :

<table>
<thead>
<tr>
<th>area (mm**2)</th>
<th>grains</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.820E-2</td>
<td>1</td>
</tr>
<tr>
<td>5.859E-2</td>
<td>1</td>
</tr>
<tr>
<td>5.937E-2</td>
<td>1</td>
</tr>
<tr>
<td>5.976E-2</td>
<td>1</td>
</tr>
<tr>
<td>6.093E-2</td>
<td>1</td>
</tr>
<tr>
<td>6.757E-2</td>
<td>2</td>
</tr>
<tr>
<td>7.109E-2</td>
<td>2</td>
</tr>
<tr>
<td>8.203E-2</td>
<td>1</td>
</tr>
<tr>
<td>8.242E-2</td>
<td>1</td>
</tr>
<tr>
<td>8.750E-2</td>
<td>2</td>
</tr>
<tr>
<td>9.023E-2</td>
<td>1</td>
</tr>
<tr>
<td>9.179E-2</td>
<td>1</td>
</tr>
<tr>
<td>9.296E-2</td>
<td>1</td>
</tr>
<tr>
<td>9.765E-2</td>
<td>2</td>
</tr>
<tr>
<td>1.113E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.175E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.199E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.238E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.242E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.328E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.410E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.496E-1</td>
<td>1</td>
</tr>
</tbody>
</table>
Table C-5 Continued

<table>
<thead>
<tr>
<th>Value</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.679E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.781E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.828E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.875E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.945E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.960E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.062E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.117E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.351E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.523E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.554E-1</td>
<td>1</td>
</tr>
<tr>
<td>3.449E-1</td>
<td>1</td>
</tr>
<tr>
<td>3.601E-1</td>
<td>1</td>
</tr>
<tr>
<td>4.238E-1</td>
<td>1</td>
</tr>
<tr>
<td>4.308E-1</td>
<td>1</td>
</tr>
<tr>
<td>4.394E-1</td>
<td>1</td>
</tr>
<tr>
<td>5.293E-1</td>
<td>1</td>
</tr>
<tr>
<td>5.773E-1</td>
<td>1</td>
</tr>
<tr>
<td>6.076E-1</td>
<td>1</td>
</tr>
<tr>
<td>6.726E-1</td>
<td>1</td>
</tr>
<tr>
<td>7.265E-1</td>
<td>1</td>
</tr>
<tr>
<td>9.078E-1</td>
<td>1</td>
</tr>
<tr>
<td>9.316E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.114E0</td>
<td>1</td>
</tr>
<tr>
<td>1.552E0</td>
<td>1</td>
</tr>
</tbody>
</table>

The average area per grain: 
2.784E-1 mm**2

The standard deviation of the distribution curve: 
3.091E-1 mm**2

The average grains per mm**2: 
3.591253 grains
Table C-6 System output data for grain structure shown in Figure 22

<table>
<thead>
<tr>
<th>The area of processed picture:</th>
<th>20.000000 mm**2</th>
</tr>
</thead>
<tbody>
<tr>
<td>The grains of size less than</td>
<td></td>
</tr>
<tr>
<td>0.039063 mm**2 (100 pixels) are ignored but the loss is compensated into each output grains</td>
<td></td>
</tr>
<tr>
<td>Total number of grains : 101</td>
<td></td>
</tr>
<tr>
<td>Boundary grains : 60</td>
<td></td>
</tr>
<tr>
<td>5.096484 mm**2 (0.254824)</td>
<td></td>
</tr>
<tr>
<td>Inner grains : 41</td>
<td></td>
</tr>
<tr>
<td>13.298047 mm**2 (0.664902)</td>
<td></td>
</tr>
<tr>
<td>The grain size distribution :</td>
<td></td>
</tr>
<tr>
<td>area (mm**2)</td>
<td>grains</td>
</tr>
<tr>
<td>5.312E-2</td>
<td>1</td>
</tr>
<tr>
<td>5.742E-2</td>
<td>1</td>
</tr>
<tr>
<td>5.820E-2</td>
<td>1</td>
</tr>
<tr>
<td>6.132E-2</td>
<td>1</td>
</tr>
<tr>
<td>6.210E-2</td>
<td>1</td>
</tr>
<tr>
<td>6.523E-2</td>
<td>1</td>
</tr>
<tr>
<td>7.070E-2</td>
<td>1</td>
</tr>
<tr>
<td>7.109E-2</td>
<td>2</td>
</tr>
<tr>
<td>7.265E-2</td>
<td>1</td>
</tr>
<tr>
<td>8.085E-2</td>
<td>2</td>
</tr>
<tr>
<td>8.164E-2</td>
<td>1</td>
</tr>
<tr>
<td>9.023E-2</td>
<td>1</td>
</tr>
<tr>
<td>9.062E-2</td>
<td>1</td>
</tr>
<tr>
<td>9.435E-2</td>
<td>1</td>
</tr>
<tr>
<td>9.570E-2</td>
<td>1</td>
</tr>
<tr>
<td>9.960E-2</td>
<td>1</td>
</tr>
<tr>
<td>1.015E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.117E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.132E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.207E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.437E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.640E-1</td>
<td>1</td>
</tr>
<tr>
<td>Value</td>
<td>Count</td>
</tr>
<tr>
<td>----------</td>
<td>-------</td>
</tr>
<tr>
<td>1.945E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.093E-1</td>
<td>2</td>
</tr>
<tr>
<td>2.101E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.253E-1</td>
<td>1</td>
</tr>
<tr>
<td>3.062E-1</td>
<td>1</td>
</tr>
<tr>
<td>3.312E-1</td>
<td>1</td>
</tr>
<tr>
<td>3.390E-1</td>
<td>1</td>
</tr>
<tr>
<td>3.878E-1</td>
<td>1</td>
</tr>
<tr>
<td>4.367E-1</td>
<td>1</td>
</tr>
<tr>
<td>4.695E-1</td>
<td>1</td>
</tr>
<tr>
<td>6.386E-1</td>
<td>1</td>
</tr>
<tr>
<td>7.464E-1</td>
<td>1</td>
</tr>
<tr>
<td>9.992E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.166E0</td>
<td>1</td>
</tr>
<tr>
<td>1.354E0</td>
<td>1</td>
</tr>
</tbody>
</table>

The average area per grain: 3.243E-1 mm**2

The standard deviation of the distribution curve: 5.230E-1 mm**2

The average grains per mm**2: 3.083160 grains
Table C-7 Grain size measurements output for the grain structure shown in Figure 16 (without the operator's intervention)

- The area of processed picture: 20.000000 mm**2

The grains of size less than 0.039063 mm**2 (100 pixels) are ignored but the loss is compensated into each output grains

Total number of grains: 108

Boundary grains: 39
- 2.003516 mm**2 (0.100176)

Inner grains: 69
- 17.135547 mm**2 (0.856777)

The grain size distribution:

<table>
<thead>
<tr>
<th>Area (mm**2)</th>
<th>Grains</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.742E-2</td>
<td>1</td>
</tr>
<tr>
<td>5.859E-2</td>
<td>2</td>
</tr>
<tr>
<td>6.093E-2</td>
<td>2</td>
</tr>
<tr>
<td>6.132E-2</td>
<td>1</td>
</tr>
<tr>
<td>6.250E-2</td>
<td>1</td>
</tr>
<tr>
<td>6.328E-2</td>
<td>1</td>
</tr>
<tr>
<td>6.367E-2</td>
<td>1</td>
</tr>
<tr>
<td>6.484E-2</td>
<td>1</td>
</tr>
<tr>
<td>6.953E-2</td>
<td>1</td>
</tr>
<tr>
<td>5.992E-2</td>
<td>1</td>
</tr>
<tr>
<td>7.031E-2</td>
<td>1</td>
</tr>
<tr>
<td>7.109E-2</td>
<td>1</td>
</tr>
<tr>
<td>7.382E-2</td>
<td>2</td>
</tr>
<tr>
<td>7.500E-2</td>
<td>2</td>
</tr>
<tr>
<td>8.281E-2</td>
<td>1</td>
</tr>
<tr>
<td>8.359E-2</td>
<td>1</td>
</tr>
<tr>
<td>8.398E-2</td>
<td>1</td>
</tr>
<tr>
<td>9.257E-2</td>
<td>1</td>
</tr>
<tr>
<td>9.296E-2</td>
<td>1</td>
</tr>
<tr>
<td>9.414E-2</td>
<td>2</td>
</tr>
<tr>
<td>1.011E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.015E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.043E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.050E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.089E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.128E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.140E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.234E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.304E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.308E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.339E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.347E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.355E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.418E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.605E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.613E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.707E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.875E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.000E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.085E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.117E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.593E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.710E-1</td>
<td>1</td>
</tr>
<tr>
<td>2.984E-1</td>
<td>1</td>
</tr>
<tr>
<td>3.046E-1</td>
<td>1</td>
</tr>
<tr>
<td>3.203E-1</td>
<td>1</td>
</tr>
<tr>
<td>3.218E-1</td>
<td>1</td>
</tr>
<tr>
<td>3.238E-1</td>
<td>1</td>
</tr>
<tr>
<td>3.406E-1</td>
<td>1</td>
</tr>
<tr>
<td>3.820E-1</td>
<td>1</td>
</tr>
<tr>
<td>3.918E-1</td>
<td>1</td>
</tr>
<tr>
<td>4.265E-1</td>
<td>1</td>
</tr>
<tr>
<td>4.933E-1</td>
<td>1</td>
</tr>
<tr>
<td>5.250E-1</td>
<td>1</td>
</tr>
<tr>
<td>5.718E-1</td>
<td>1</td>
</tr>
<tr>
<td>6.054E-1</td>
<td>1</td>
</tr>
<tr>
<td>6.703E-1</td>
<td>1</td>
</tr>
<tr>
<td>6.812E-1</td>
<td>1</td>
</tr>
<tr>
<td>6.933E-1</td>
<td>1</td>
</tr>
<tr>
<td>7.078E-1</td>
<td>1</td>
</tr>
<tr>
<td>7.398E-1</td>
<td>1</td>
</tr>
<tr>
<td>8.332E-1</td>
<td>1</td>
</tr>
<tr>
<td>1.134E0</td>
<td>1</td>
</tr>
<tr>
<td>1.156E0</td>
<td>1</td>
</tr>
</tbody>
</table>

The average area per grain:
2.483E-1 mm**2

The standard deviation of the distribution curve:
2.563E-1 mm**2

The average grains per mm**2:
4.026717 grains
### Table C-8 Wireframe representation data file of Figure 14(b)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>1</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>2</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>3</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>4</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>5</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>6</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>7</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>8</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>9</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>10</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>11</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>12</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>13</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>14</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>15</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>16</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>17</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>18</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>19</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
</tr>
</tbody>
</table>
Figure C-1 Smoothed histogram of Figure 8 and the local minimums selected by the program.
Figure C-2 Smoothed histogram of Figure 10 and the local minimums selected by the program.
Appendix D

Program Listing

This section includes the software implementations of the digital image processing techniques and the grain size measuring program discussed in the previous sections. Table D-1 provides a summary of the programs.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Language</th>
<th>Machine</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>capture</td>
<td>Program for activating the Chorus 6-bit digitizer and storing the digitized picture onto disk in unpacked format</td>
<td>basic</td>
<td>IBM-PC</td>
<td>97</td>
</tr>
<tr>
<td>display</td>
<td>Program for displaying the unpacked digital picture</td>
<td>basic</td>
<td>IBM-PC</td>
<td>99</td>
</tr>
<tr>
<td>histgrm</td>
<td>Program for histogram smoothing and local minimums selecting</td>
<td>basic</td>
<td>IBM-PC</td>
<td>100</td>
</tr>
<tr>
<td>histc</td>
<td>Histogram equalization program</td>
<td>C</td>
<td>IBM-PC</td>
<td>104</td>
</tr>
<tr>
<td>hiemfsis</td>
<td>Spatial domain Laplacian high emphasis filtering program</td>
<td>C</td>
<td>DEC PDP 11/55</td>
<td>106</td>
</tr>
<tr>
<td>filter</td>
<td>Frequency domain filtering program</td>
<td>C</td>
<td>DEC PDP 11/55</td>
<td>108</td>
</tr>
<tr>
<td>homo_pass1</td>
<td>Homomorphic filtering program</td>
<td>C</td>
<td>DEC PDP 11/55</td>
<td>113</td>
</tr>
<tr>
<td>homo_pass2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>smooth</td>
<td>Spatial domain averaging program</td>
<td>C</td>
<td>IBM-PC</td>
<td>123</td>
</tr>
<tr>
<td>wiener</td>
<td>Wiener filtering program</td>
<td>C</td>
<td>DEC PDP 11/55</td>
<td>124</td>
</tr>
<tr>
<td>wiener_pass1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wiener_pass2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fd</td>
<td>First derivative edge detection program</td>
<td>C</td>
<td>IBM-PC</td>
<td>132</td>
</tr>
<tr>
<td>sobel</td>
<td>Sobel edge detection program</td>
<td>C</td>
<td>IBM-PC</td>
<td>134</td>
</tr>
<tr>
<td>template</td>
<td>Template matching program</td>
<td>C</td>
<td>DEC PDP 11/55</td>
<td>137</td>
</tr>
<tr>
<td>segment</td>
<td>Histogram segmentation program</td>
<td>C</td>
<td>IBM-PC</td>
<td>141</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------</td>
<td>---</td>
<td>--------</td>
<td>-----</td>
</tr>
<tr>
<td>size</td>
<td>Grain size measuring program</td>
<td>C</td>
<td>IBM-PC</td>
<td>142</td>
</tr>
<tr>
<td>fft</td>
<td>2-D FFT program</td>
<td>C</td>
<td>DEC PDP 11/55</td>
<td>149</td>
</tr>
<tr>
<td>flipreal</td>
<td>Subroutine used in the fft program</td>
<td>C</td>
<td>DEC PDP 11/55</td>
<td>152</td>
</tr>
</tbody>
</table>
REM PROGRAM CAPTURE.BAS
20 REM
30 REM This program activates the Chorus 6-bit digitizer
40 REM and stores the digitized picture onto disk in unpacked format
50 REM
60 REM written by Chris K. Wu in IBM PC BASICA
70 REM
80 REM download subroutines
90 REM
100 KEY OFF:DEF SEG = 4H2AF4
110 BADDR=4HAE;BAFRAME=4H2F;BCALIB=4HD9;BCURMOV=4H47F;BDDMA=4H4C
120 BDFRAME=4H4D;BDPIO=6H4F;BEYERAD=6H111;BEYWRITE=6H87;BPRNSCRN=6H35D
130 BSETCRT=6HFA;BSETMOD=6H83;BSETREG=6H3D1:BTHROT=6H50:BWEIGHT=6H48F
140 IX%=0;TLERR%=0;ERR%=0:BDDMA=4H341;BMHTDSP=6H1FD;BUWRITE=6H1C2
150 BLOAD "pceye10.1rd",0
160 DEF SEG = 6H1C09
170 XTRC%=0;IOMODE%=0;METY%=5
180 CLS
190 REM
200 REM parameters setup for the type of camera used
210 REM
220 BEEP:PRINT "camera type 
230 PRINT ": 1) noninterlaced"
240 PRINT ": 2) interlaced"
250 PRINT
260 INPUT "your choice (1/2)";ANS$ 
270 IF (ANS$="2") GOTO 300 
280 IF (ANS$="1") GOTO 330 
290 GOTO 180
300 INTER%=0;HOFF%=32;VOFF%=68
310 LEFT%=0;HPEL%=440;TOP%=0;VPEL%=400 
320 GOTO 350
330 INTER%=0;HOFF%=43;VOFF%=7
340 LEFT%=0;HPEL%=440;TOP%=0;VPEL%=338
350 CALL BSETMOD((INTER%,XTRC%,IOMODE%,METY%,ERR%))
360 PBASE%=6H10;GBASE%=6H3D0;DMEM%=6HA000;MMEM%=6H3000
370 CALL BADDR(PBASE%,GBASE%,DMEM%,MMEM%,ERR%)
380 CALL BAFRAME(HOFF%,VOFF%,IERR%)
390 CALL BDFRAME(LEFT%,HPEL%,TOP%,VPEL%,IERR%)
400 TRLTL%=14
410 CALL BTHROT(TRLTL%,IERR%)
420 OFFSET%=1;CHANNEL%=48
430 CALL BSETREG(OFFSET%,CHANNEL%,IERR%)
440 TYPE%=0
450 MODE%=0:CALL BSETCRT(MODE%)
460 COUNTER=0
470 BEEP
480 CLS
490 PRINT "process type 
500 PRINT ": 1) still frame with data saved"
510 PRINT ": 2) continuous picture display (3 pictures)"
520 INPUT "your choice (1/2)";ANS$ 
530 IF (ANS$="2") THEN PROTY%=1 :GOTO 410
540 IF (ANS$="1") THEN PROTY%=0 :GOTO 560
550 GOTO 470
560 INPUT "filename ";FYLE$ 
570 FILE=FYLE$".img"
580 REM
590 REM input the quantisation scale
600 REM
610 INPUT "floor gray level "; BLACK%
620 INPUT "ceiling gray level "; WHITE%
630 CALL BCALIB(BLACK%, WHITE%, IERR%)
640 MODE%= 15: CALL BSETCRT(MODE%)
650 CALL BM DMA(TYPE%, IERR%)
660 COUNT = COUNT + 1
670 CALL BMH TDisp(TYPE%)
680 IF (PROTYE%= 1 AND COUNT < 5) GOTO 450
690 IF (COUNT = 5) GOTO 450
700 TYPE%= 2; IERR%= 0
710 CALL BUIWRIT(FLs, TYPE%, IERR%)
720 BEEP
730 A$=INKEY$ 
740 IF A$="" THEN 730
750 MODE%= 0: CALL BSETCRT(MODE%)
760 INPUT "continue "; ANS$
770 IF (ANS$ = "y") GOTO 450
780 END
10 REM PROGRAM DISPLAY.BAS
20 REM
30 REM This program displays the unpacked digital pictures captured by Chorus 6-bit digitiser
40 REM written by Chris K. Wu in IBM PC BASICA
50 REM
60 KEY OFF:DEF SEG = &H2AF4
80 BDFRAME=&H4:BDPDI=&H42F:BEYREAD=&MH111:BEYWRITE=&H167:BPRNSCRN=&H3D
90 BSETCR%=&HFA:BSETMOD=&H83:BSETREG=&HD3D1:BTHROT%=&H50:BWEIGHT=&H48F
100 IX%=0:TBERR%=0:IRM%=0:BDMA=&H341:BMMDISP=&H1FD:BUWRITE=&H1C2
110 BIUREAD=&H1C4
120 BLOAD "pcexrp10.10",0
130 DEF SEG = &H2CD9
140 INTERVAL%=1:XRNG%=0:IMODE%=0:MEMTYPE%=5
150 CLS:BEER:PRINT "picture size"
160 PRINT " 1)half size"
170 PRINT " 2)full size"
180 INPUT "Your choice (1/2):";ANS$
190 IF ANS$="1" GOTO 240
200 IF ANS$="2" GOTO 270
210 GOTO 170
220 GOTO 220
230 GOTO 170
240 INTERVAL%=1:HOFF%=48:VOFF%=2
250 LEFT%=0:HPEL%=640:TOP%=0:VPEL%=238
260 GOTO 290
270 INTERVAL%=0:HOFF%=32:VOFF%=68
280 LEFT%=0:HPEL%=640:TOP%=0:VPEL%=400
290 CALL BSSETMOD(INTER%,XRNG%,IMODE%,MEMTYPE%,ERR%)
300 PBASE%=&HD310:GBASE%=&HD3D0:DMEM%=&HA000:MEM%=&H3D00
310 CALL BADDR(PSBASE%,GBASE%,DMEM%,MEM%,ERR%)
320 CALL BAFRAME(HOFF%,VOFF%,ERR%)
330 CALL BDFRAME(LEFT%,HPEL%,TOP%,VPEL%,ERR%)
340 TRTL%=16
350 CALL BTHROT(LEFT%,ERR%)
360 OFFSET%=1:CHANNEL%=48
370 CALL BSSETREG(OFFSET%,CHANNEL%,ERR%)
380 INPUT "picture filename ";FNAME$
390 BEEP:TYPE%=2:ERR%=0
400 CALL BUREAD(FNAME$,TYPE%,ERR%)
410 FOR I=1 TO 2000:NEXT I
420 COUNT=0
430 MODE%=15:CALL BSSETCRT(MODE%)
440 TYPE%=0
450 CALL BMMDISP(TYPE%)
460 FOR I=1 TO 5000:NEXT I
470 BEEP
480 A$=INKEY$
490 IF A$="" GOTO 490
500 MODE%=0:CALL BSSETCRT(MODE%)
510 GOTO 380
10 REM PROGRAM HISTGM.BAS
20 REM
30 REM This program smooths the histogram of the input picture
40 REM and searches the local minimum of the smoothed histogram
50 REM written by Chris K. Wu in IBM PC BASICA
60 REM
70 DIM GY(64),SM(64),FD(64)
80 FOR I=0 TO 63
90 GY(I)=0
100 NEXT I
110 KEY OFF:DEF SEG = &H2E4F
130 BDFRAME=$64H:BDPO=$62F:BEYERED=$6111:BEYERWRITE=$617:BPNSRCRN=$635D
140 BSETCRT=6HA: BSETMOD=6H83: BSETREG=6H121:BTHROT=6H30: BWEIGHT=6H48F
150 IERR%=0: BERR%=0: HDMBA=$6341: BMHTDISP=$61FD: BIUIREAD=$614C
160 BUREAD=$614C
170 BLOAD "peyea10.10",0
180 DEF SEG = &H2CD9
190 INTER%=1: XTRG%=0: IOMODE%=0: MENTYP%=5
200 INTER%=1: HOFF%=48: VOFF%=2
210 LEFT%=0: HPEL%=0: TOP%=0: VPEL%=238
220 CALL BSETMOD(INTER%, XTRG%, IOMODE%, MENTYP%, IERR%)
230 PA%=$6310: GBASE%=&H300: DMEN%=&HA000: MMEN%=&H3000
240 CALL BADDR(PBASE%, GBASE%, DMEN%, MMEN%, IERR%)
250 CALL BAFRAME(HOFF%, VOFF%, IERR%)
260 CALL BDFRAME(LEFT%, HPEL%, TOP%, VPEL%, IERR%)
270 TRTL%=16
280 CALL BTHROTL(TRTL%, IERR%)
290 OFFSET%=1: CHANNEL%=46
300 CALL BSETREG(OFFSET%, CHANNEL%, IERR%)
310 INPUT "picture filename "; FYLE$
320 PRINT " restriction: 102 < row < 306 "
330 PRINT " 2 < col < 384 "
340 INPUT " picture size (row, col) "; ROW, COL
350 BEEP: TYPE%=2: IERR%=0
360 CALL BUREAD(FYLE$, TYPE%, IERR%)
370 FOR I=1 TO 500:NEXT I
380 COUNT=0
390 MODE%=15: CALL BSETCRT(MODE%)
400 TYPE%=0
410 CALL BMHTDISP(TYPE%)
420 FOR I=1 TO 500:NEXT I
430 BEEP
440 MODE%=0: CALL BSETCRT(MODE%)
450 REM
460 REM seek the pixel value and print the value
470 REM and the pixel position on the CRT
480 REM
490 CLS
500 LOCATE 11,26,0
510 PRINT " pixel position grayvalue "
520 LOCATE 12,28: PRINT " row col "
530 X=27: Y=33: G=48: JSTART=&440-COL
540 DEF SEG=$6H3000
550 FOR I=0 TO 101 STEP 4
560 LOCATE 13,X: PRINT I
570 FOR J=0 TO (COL-1) STEP 4
580 LOCATE 13,Y: PRINT " ";LOCATE 13,Y: PRINT J
590 V%=PEEK(JSTART+J+$440*1)
600 LOCATE 13,G: PRINT " ";LOCATE 13,G: PRINT V%
610 GY(V%) = GY(V%) + 1
620 NEXT J
630 NEXT I
640 JSTART=384-COL
650 DEF SEG=4H4000
660 IF ROW<204 THEN ISTOP=ROW-1 ELSE ISTOP=203
670 FOR I=1 TO ISTOP STEP 4
680 LOCATE 13,X:PRINT I
690 FOR J=0 TO (COL-I) STEP 4
700 LOCATE 13,Y:PRINT "":LOCATE 13,Y:PRINT J
710 V%=PEEK(JSTART+J+440*(I-102))
720 LOCATE 13,G:PRINT "":LOCATE 13,G:PRINT V%
730 GY(V%) = GY(V%) + 1
740 NEXT J
750 NEXT I
760 IF ROW<205 GOTO 880
770 JSTART=128+640-COL
780 DEF SEG=4H5000
790 FOR I=205 TO ROW-1 STEP 4
800 LOCATE 13,X:PRINT I
810 FOR J=0 TO (COL-I) STEP 4
820 LOCATE 13,Y:PRINT "":LOCATE 13,Y:PRINT J
830 V%=PEEK(JSTART+J+440*(I-205))
840 LOCATE 13,G:PRINT "":LOCATE 13,G:PRINT V%
850 GY(V%) = GY(V%) + 1
860 NEXT J
870 NEXT I
880 CLS
890 SCREEN 2
900 REM
910 REM plot the histogram on the CRT
920 REM
930 LOCATE 1,3:PRINT "1"
940 LOCATE 20,3:PRINT "0"
950 FOR I=1 TO 15
960 LOCATE 22,44+4*I:PRINT 4*I;
970 NEXT I
980 LOCATE 22,5:PRINT "9"
990 LOCATE 22,69:PRINT "63"
1000 LINE (33,0)-(33,140)
1010 LINE (32,150)-(327,142)
1020 LOCATE 25,3:PRINT "-X- GRAYLEVELS, -Y- FREQUENCY OF OCCURRENCE"
1030 TOTL=ROW-COL/128
1040 FOR I=0 TO 63
1050 PCT=GY(I)/TOTL
1060 LINE (35+8*I,160)-(35+8*I,160-160*PCT)
1070 NEXT I
2000 CLS
2010 SM(0)=(GY(0)+GY(1))/2
2020 FOR I=1 TO 62
2030 SM(I)=(GY(I-1)+GY(I)+GY(I+1))/3
2040 NEXT I
2050 SM(63)=(GY(62)+GY(63))/2
2060 CLS
2070 REM
2080 REM plot the histogram on the CRT
2090 REM
2100 LOCATE 1,3:PRINT "1"
2110 LOCATE 20,3:PRINT "0"
2120 FOR I=1 TO 15
2130 LOCATE 22,4; PRINT 4*I;
2140 NEXT I
2150 LOCATE 22,5; PRINT "0"
2160 LOCATE 22,6; PRINT "43"
2170 LINE (33,0)-(33,140)
2180 LINE (33,160)-(539,160)
2190 LOCATE 25,5; PRINT "-X- GRAYLEVELS, -Y- FREQUENCY OF OCCURENCE";
2200 FOR I=0 TO 63
2210 PCT=SM(I)/TOTL
2220 LINE (35+8*I,160)-(35+8*I,160-160*PCT)
2230 NEXT I
3000 CLS
3010 MIN=100
3020 MAX=0
3030 FOR I=1 TO 63
3040 FD(I)=SM(I)-SM(I-1)
3050 IF (MAX ( FD(I) ) THEN MAX=FD(I)
3060 IF ( MIN ( FD(I) ) THEN MIN=FD(I): PMIN=I
3070 IF (ABS(MIN)<MAX) THEN SCALE=ABS(MIN) ELSE SCALE=MAX
3080 NEXT I
3090 LOCATE 1,1; PRINT MAX;
3100 LOCATE 20,1; PRINT MIN;
3110 FOR I=1 TO 15
3120 LOCATE 22,4+4*I; PRINT 4*I;
3130 NEXT I
3140 LOCATE 22,5; PRINT "0"
3150 LOCATE 22,6; PRINT "43"
3160 LINE (33,0)-(33,160)
3170 LINE (33,80)-(539,80)
3180 LOCATE 25,5; PRINT "-X- GRAYLEVELS, -Y- GRADIENT";
3190 FOR I=1 TO 63
3200 IF (FD(I) < 0) GOTO 3240
3210 LEND=(FD(I)/SCALE)*80
3220 LINE (35+8*I,80)-(35+8*I,80-LEND)
3230 GOTO 3240
3240 LEND=ABS(FD(I)/SCALE)*80
3250 LINE (35+8*I,80)-(35+8*I,80+LEND)
3260 NEXT I
4000 CLS
4010 FOR I=PMIN TO 62
4020 IF (FD(I+1) > 0) THEN PMIN=I: I=42
4030 NEXT I
4040 MARK(0)=PMIN
4050 J=1
4060 CP=PMIN
4070 FOR I=PMIN+1 TO 62
4080 IF ((FD(I) < 0) AND (FD(I+1) > 0) AND ((I-CP) = 4)) GOTO 4100
4090 GOTO 4170
4100 FOR K=1 TO 3 STEP -1
4110 IF (FD(K) <= 0) THEN GOTO 4120 ELSE GOTO 4170
4120 NEXT K
4130 MARK(J)=I
4140 CP=I
4150 J=J+1
4160 IF (J = 4 ) GOTO 4300
4170 NEXT I
4180 CP=PMIN
4190 FOR I=PMIN+1 TO 62 STEP -1
4200 IF ((FD(I) < 0) AND (FD(I+1) > 0) AND ((CP-I) = 4)) GOTO 4220
4210 GOTO 4270
4220 FOR K=I-1 TO I-3 STEP -1
4230 IF (FD(K) <= 0) THEN GOTO 4240 ELSE GOTO 4290
4240 NEXT K
4250 MARK(J)=I
4260 CP=1
4270 J=J+1
4280 IF ( J = 4 ) GOTO 4300
4290 NEXT I
4300 FOR I = 0 TO J-1
4310 PRINT "MARK(";I;")= ";MARK(I)
4320 NEXT I
/* PROGRAM HISTC -- Histogram equalization program.
   Nonlinear histogram correction for image enhancement.

*/
#include <stdio.h>
#include <math.h>
FILE *fopen(), *rfp, *wfp;
int row, col, lt[64], err;
main(ac, av)
    int ac, char *av[];
{
    int i, j, nt, inbuf[640],ans;
    float x, den, num;
    if(ac!=5) {
        printf("usage: histc filein fileout row col \n");
        printf("nonlinear histogram correction for image enhancement \n");
        printf("filein -- input filename\n");
        printf("fileout -- output filename\n");
        printf("row -- # of rows\n");
        printf("col -- # of columns\n");
        exit();
    }
    row=atoi(av[3]);
    col=atoi(av[4]);
    repeat:
    printf("\n\n");
    printf("type of process ?\n");
    printf(" 1 gamma correction\n");
    printf(" 2 integration\n");
    printf(" your choice ? ");
    scanf("%d", &ans);
    if(ans!=1 && ans!=2) goto repeat;
    switch (ans) {
    case 1:
        printf("x = ");
        scanf("%f", &x);
        deno=(float)exp((double)i/lt)-1;
        for (i=0;i<64;i++) {
            nume=(float)exp((double)nume)-1;
            if(i==lt){
                printf("\n\n");
            }
        }
        break;
    case 2:
        compav[3];
        break;
    }
    rfp=fopen(av[1], "r");
    wfp=fopen(av[2], "w");
    for(i=0;i<row;i++) {
        err=fread(inbuf, 2, col, rfp);
    }
    for(j=0;j<col;j++) {
if (inbuf[j]==0) continue;
    ni=inbuf[j];
inbuf[j]=nl[nt];
}
err=write(inbuf,2,col,wpf);
}
fclose(rfp);
fclose(wpf);
exit();
}
comp(rast)
char *rast;
{
    int i,j,k,snbuf[640],buf[64];
    float scale;
    double rad;
    scale=63./(float)(row*col);
    rfp=fopen(rast,"r");
    for(i=0;i<64;i++) buf[i]=0;
    for(i=0;i<row;i++)
    {
        err=fread(snbuf,2,col,rfp);
        for(j=0;j<col;j++)
        {
            k=snbuf[j];
            buff[k]++;  
        }
    }
    for(i=0;i<64;i++)
    {
        for(j=0;j<col;j++)
        {
            ltt[i]=buff[i];
            ltt0[i]=buff[0];
            ltt[i]=ltt[i]-1+buff[i];
            ltt[i]=(int)((float)ltt[i]*scale);
            ltt[i]=ltt[i]+1;
            ltt[i]-=(int)((float)ltt[i]*scale);
        }
    }
    fclose(rfp);
    }
/*
PROGRAM HIEMFS1 - this routine applies the technique of a subtraction
combination of a 'raster' and its Laplacian to emphasize the high
frequency component of the 'raster'.
compile with `/usr/PPS/lib/ppslib` and `/usr/PPS/lib/util.a
chris i. wu, Apr. '85
*/

#include "<usr/PPS/include/pps.h>
#include <stdio.h>
#define ROWS '3

int **row;

main(argc, argv)
int argc; char **argv;
{

int *reply, i, j, sn, j;
int **arralloc(), *temp, *outbuf, COLS, *tstalloc();
char *mod[100];
double *sum=0.0;

if(argc != '3') {
    printf("Usage: %s 'filein' 'fileout' \n", argv[0]);
    printf("this routine emphasizes the high frequency component\n");
    printf("of a 'raster' file. \n\n");
    printf("'filein' -- the input PPS 'raster' 'filename'\n");
    printf("'fileout' -- the output 'processed' 'filename'\n");
    exit();
}

ppsinit(argc, argv);
openfile(0, argv[1], RD);
ppssreport(0);

COLS=argv[1], ncols;
outbuf=tstalloc(COLS*2);
row=arralloc(ROWS, COLS);

createfile(1, argv[2], 0);
ancestor(1, 0);
sprintf(mod,"Enhanced 'with' a 'subtraction' combination of 'file' and its 'Laplacian.'");
strftime(t[i], header, mod);
hflush(1);
findrow(1, 0);

/*Initialize 'rows*/
for(i=0; i<ROWS; i++)

    if((reply->readrow(0, row[i]))!=COLS*2){
        printf(stderr,"read 'error' when initialize 'rows'\n");
        exit();
    }
if '((reply=writerow(1, row[0])!="COLS\#2") '{
    fprintf(stderr,"write error on row \%d, \%d bytes written\n", i, reply);
    exit();
}'

/* Now 'start 'looping 'through 'data*/
for '((i=1; i<COLS-1; i++) '{
    for '((j=1; j<COLS-1; j++) '{
        for '((i1=0; i1<COLS; i1++) '{
            sum=sum+(double)row[i1][j];
        }
    }
    for '((i1=0; i1<COLS; i1++) '{
        if '((i1 == 0)' '{
            sum=sn=0.0;
        }
    }

    sum=sum/5.0;
    sum=(double)row[1][j]-sum;
    sum=(double)row[1][j]+5.0*sum;
    outbuf[j]=(int)sum;
    sum=0.0;
}
}

if '((reply=writerow(1, outbuf))!="COLS\#2") '{
    fprintf(stderr,"write error on row \%d, \%d bytes written\n", i, reply);
    exit();
}'

if '((i<COLS-2)' '{
    temp=row[0];
    for '((i1=0; i1<COLS; i1++) '{
        row[i1]=row[i1+1];
        row[2]=temp;
    }
}

if '((reply=readrow(0, row[2]))!="COLS\#2") '{
    fprintf(stderr,"read error on row \%d\n", i);
    exit();
}'

write_row(1, row[2]);
pps_end();
exit();
}
Apr 23 22:34 '86 'filter.c 'Page '1

/* PROGRAM 'FILTER' -- This program removes noise by using 'low pass'
filter or enhances 'edges' by using 'high emphasis'
filter.

'Compiled with /usr/PPS/lib/ppslib' and '/usr/PPS/lib/unutil.a
'Chris H. Wu, Apr. '85
*/

#include "usr/PPS/include/ppsl.h"
#include <math.h>

int row, col;
char *tsmalloc();

main(ac, av)

int ac; 'char *av;
{

int i, j, fp1, fp2, opt1, opt2, opt3, rax[5];
int radius;
char *file[50], *lin[50], *out[50], cmd[50];

if (ac!=9) { 'c

..... printf("Usage: %s 'pdata 'redata 'imdata 'fileout 'opt1 'opt2 'xsize 'ysize
", av[0]);
..... printf("This routine enhances 'image' by employing 'filtering'");
..... printf("techniques\n");
..... printf("pdata " 'power 'spectrum 'data " 'filename\n");
..... printf("redata " 'real 'data " 'filename\n");
..... printf("imdata " 'imaginary 'data " 'filename\n");
..... printf("fileout " 'output 'filename\n");
..... printf("opt1 " 'O-' 'low pass 'filter\n");
..... printf("opt2 " 'O-' 'butterworth 'filter\n");
..... printf("opt3 " 'O-' 'exponential 'filter\n");
..... printf("xsize " 'num. 'of 'rows\n");
..... printf("ysize " 'num. 'of 'columns\n");
..... exit('0);

..... }

ppslinit(ac, av);
fp1=fopen(av[1], 'O');
read(fp1, rax, 10);
close(fp1);
opt1=stoi(av[3]);
opt2=stoi(av[6]);
row=stoi(av[7]);
col=stoi(av[8]);
for (i=0; i<5; i++) printf("energy = \%f \r = \%d\n", 5. * (float)i, rax[i]);
printf("radius = \%d\n", radius);
scanf("%d", &radius);
printf("radius = \%d", radius);
printf(file, "filter\n");

}
Apr '23 22: 34 '86 'Filter.c 'Page 2

```
sprintf(filim, "filtrim");
sprintf(filout, "filtrek");
switch (opt2) {
    case 0: butterworth (av[2], av[3], filre, filim, radius);
    case 1: exponential (av[2], av[3], filre, filim, radius);
    break;
}
if (opt1) high (av[2], av[3], filre, filim);
printf(cmd, "ifft %s %s %s %d ", filre, filim, av[4], filout, row);
printf("%s\n", cmd);

system(cmd);
sprintf(cmd, "rm %s %s", filre, filim);
printf("%s\n", cmd);
system(cmd);
}
pp_end();
exit();
```

/* BUTWORTH -- 'butterworth 'Filtering 'subroutine '*/

butterworth (redata, imdata, ou tre, outim, rad)

int *rad;
char *redata, *imdata, *ou tre, *outim;

<

int i, j, fp1, fp2, fp3, fp4, x, yc;
float *rbuf, *ibuf, b, d;

fp1 = open (redata, 0);
fp2 = open (imdata, 0);
fp3 = creat (ou tre, 0664);
fp4 = creat (outim, 0664);
rbuf = talloc (col * 4);
ibuf = talloc (col * 4);
x = row / 2;
y = col / 2;
for (i = 0; i < row; i++) {
    read (fp1, rbuf, col * 4);
    read (fp2, ibuf, col * 4);
    for (j = 0; j < col; j++) {
        d = (float) sqrt ((double) ((x - i) * (x - i)) + (double) ((yc - j) * (yc - j)));
    }
```
b=1/(1 + std/(float)(rad*rad));

rbuf[j]=rbuf[j]+b;

ibuf[j]=ibuf[j]+b;

write(fp3, rbuf, col#4);
write(fp4, ibuf, col#4);

close();


/* EXPONENTIAL — This routine enhances image by employing exponential filtering technique */

exponential(redat, imdata, outre, outim, rad)

int rad;
char *redat, *imdata, *outre, *outim;

{
int i, j, fp1, fp2, fp3, fp4, xc, yc;
float *rbuf, *ibuf, b, d;

fp1=open(redat, O);
fp2=open(imdata, O);
fp3=creat(outre, 0644);
fp4=creat(outim, 0644);
rbuf=malloc(col#4);
ibuf=malloc(col#4);
xc=row/2;
yc=col/2;
for (i=0; i<row; i++) {

read(fp1, rbuf, col#4);
read(fp2, ibuf, col#4);
for (j=0; j<col; j++) {

    d=(float)sqrt((double)((xc-i)*(xc-i))+(double)((yc-j)*(yc-j)));
    b=exp(-0.693147*(d/(float)rad)*(d/(float)rad));
    rbuf[j]=rbuf[j]*b;
    ibuf[j]=ibuf[j]*b;
}

write(fp3, rbuf, col#4);
write(fp4, ibuf, col#4);
close();

/\ 'HIGH' -- 'high pass filtering '*/
high(r,ida,ota,otim)
char *rdata, *idata, *ota, *otim;
{
int i, j, fp1, fp2, fp3, fp4, fp5, fp6;
float *rrbuf, *ibuf, *dbuf, *dibuf; 'char cmd[100], tp1[50], tp2[50];

fp1=fopen(rdata, "r");
fp2=fopen(idata, "r");
p3=fopen(ota, "r");
fp4=fopen(otim, "r");
rrbuf=tstalloc(c1e4);
ibuf=tstalloc(c1e4);
dbuf=tstalloc(c1e4);
dibuf=tstalloc(c1e4);
sprintf(tp1, "tpf11");
printf(tp2, "tpf12");
fp5=creat(tp1, 0644);
fp6=creat(tp2, 0644);
for (i=0; i<rown, i++) {

read(fp1, drbuf, c1e4);
read(fp2, dbuf, c1e4);
read(fp3, rbuf, c1e4);
read(fp4, ibuf, c1e4);
for (j=0; j<c1e4, j++) {

rrbuf[j]=1.5*drbuf[j]-rrbuf[j-1];
ibuf[j]=1.5*ibuf[j]-ibuf[j-1];

}
write(fp5, rbuf, c1e4);
write(fp6, ibuf, c1e4);
}

close();
unlink(ota);
sprintf(cmd, "mv %s %s", tp1, ota);
printf("%s\n", cmd);
system(cmd);
unlink(otim);
sprintf(cmd, "mv %s %s", tp2, otim);
printf("%s\n", cmd);
system(cmd);
}
//
// PROGRAM HOMO_PAS1 — this routine generates the logarithmic
// Fourier Transform and the power spectrum from the
// input raster for the use of Homo_pas2.
// compile with `/usr/PPS/lib/ppslib`, `/usr/PPS/lib/util.a`
// `lib/libm.a` and `/lib/libmath.a`

chris k. wu, Apr. '85
*/

#include "<usr/PPS/include/pps.h>
#include `<stdio.h>
#include `<math.h>
#define PMODE 0644
#define BUFSIZE 128

int row, col, *integer;
char *cmd[1000];

main(ac, av)

int *ac, char **av;
{
char *filere[50], fileim[50], file[50];

if *(ac) == 3

printf("Usage: %s rastin 'fileout'\n", av[0]);
printf("This routine generates the logarithmic F. T 'and\n");
printf("the power spectrum from the input raster for \n");
printf("the use of Homo_pas2\n");
printf("rastin' input raster 'filename')

printf("fileout' output power spectrum 'filename')

exit();

}

ppsinit(ac, av);

sprintf(filere, "redata");
sprintf(fileim, "imdata");
sprintf(file, "radius.h");

f(t(av[1], filere, fileim);
ftp(filere, fileim, av[2], file);

pps_end();
exit();

}

fft(raster, filere, fileim)
char *raster, *filere, *fileim;
May '10 12:17 '1985 'homo_pasi.c 'Page 2

{

char *tmpf[50], tmpf2[50];
int reply, i, j, bufsize, n, m, nt, fp1, fp2, fp3, fp4;
int temp[BUFSIZE];
float rebuff[BUFSIZE], imbuf[BUFSIZE];

openfile(0, raster, RD);
ppreport(0);
fp1=creat(filen, PMODE);
fp2=creat(filen, PMODE);

if ((row=h[01].nrows) > '128 ' || (col=h[01].ncols) > '128 ') {
    printf("Input file size is restricted at maximum '128x128'\n");
    exit();
}
bufsize=col+2;

/* Perform FFT of each row*/

nt=col;
n=nt;
m=0;
while (nt>1) {
    
    nt=nt/2;
    m++;
    
}

for (i=0; i<row; i++) {
    
    if ((reply=readrow(0, temp)) != 'bufsize') {
        printf(stderr, "read error on row %d. %d bytes read\n", i, reply);
        exit();
    }
    
    for (j=0; j<col; j++) {
        
        nt=i+j;
        imbuf[j]=0;
        if (temp[i] > 0) rebuff[j]=0;
        else rebuff[j]=(float)power(-1, nt)*(float)log((double)temp[i]);
        
        fft2(rebuff, imbuf, n, m, 1);
        write(fp1, rebuff, bufsize);2);
        write(fp2, imbuf, bufsize);2);
        
    }

    close(fp1);
    close(fp2);
```c
printf(cmd, "flipreal %s %d", filere, col);
printf("%s\n", cmd);
system(cmd);
printf(cmd, "flipreal %s %d", fileim, col);
printf("%s\n", cmd);
system(cmd);

/* Perform FT of each column */

n=row;
nt=n;
m=0;
while (nt>1) {
    nt=nt/2;
    m++;
}

fp1=fopen(filere, O);
fp2=fopen(fileim, O);
printf(tmpf1, "tempr");
printf(tmpf2, "tempi");
fp3=creat(tmpf1, PMODE);
fp4=creat(tmpf2, PMODE);
for (i=0; i<col; i++) {
    read(fp1, rebuf, bufsize*2);
    read(fp2, imbuf, bufsize*2);
    fwrite(rebuf, imbuf, n, m, 1);
    write(fp3, rebuf, bufsize*2);
    write(fp4, imbuf, bufsize*2);
}
close(fp1);
close(fp2);
close(fp3);
close(fp4);
unlink(filere);
printf(cmd, "mv tempr %s", filere);
printf("%s\n", cmd);
system(cmd);
unlink(fileim);
printf(cmd, "mv tempi %s", fileim);
printf("%s\n", cmd);
system(cmd);
printf(cmd, "flipreal %s %d", filere, col);
printf("%s\n", cmd);
system(cmd);
printf(cmd, "flipreal %s %d", fileim, col);
printf("%s\n", cmd);
system(cmd);
```
May '10 '12: 17 '1985 'homo_pasi.c 'Page '4

```c
system(cmd);
}

power(x, n)
int x, n;
{
int i, p;
p=1;
for (i=1; i<=n; i++) p=p*x;
return(p);
}

ftp(filein, fileout, file)
char *filein, *fileout, *file;
{
float *rebf[BUFSIZE], imbf[BUFSIZE], tmp[BUFSIZE];
int i, j, l, n, fp1, fp2, fp3, fp4;
int xc, yc, r, rc[3];
double sum, check, cpt, arrck[3];

fp1=fopen(filein, O);
fp2=fopen(filein, O);
fp3=fopen(fileout, PMODE);
fp4=fopen(fileout, PMODE);
sum=0;
for (i=0; i<row; i++)
{
    //read(fp1, rebf, col=4);
    //read(fp2, imbf, col=4);
    //for (j=0; j<col; j++)
    //    tmp[j]=(float)sqrt((double)(rebf[j]*rebf[j]+imbf[j]*imbf[j]));
    //    sum=sum+(double)tmp[j];
    //}
    write(fp3, tmp, col=4);
    //}
    close(fp1);
    close(fp2);
    close(fp3);
    fp1=fopen(fileout, O);

    for (i=0; i<5; i++)
        arrck[i]=(.5+(double)i)*sum;
    l=0;
    cpt=arrck[l++];
    xc=row/2;
    yc=col/2;
    n=0;
```
for (i=0; i<5; i++) rc[i]=0;
for (r=0; r<Xc; r++) {
    for (i=0; i<Xc-r; i++) {
        read(fp1, rebf, col+4);
        read(fp1, rebf, col+4);
        for (j=0; j<r+1; j++) {
            if (j==0) check=check+(double)rebf[yc];
            else check=check+(double)rebf[yc+j]+(double)rebf[yc-j];
        }
        if (r==0) goto jump;
        for (i=0; i<(2*r-1); i++) {
            read(fp1, rebf, col+4);
            check=check+(double)rebf[yc+r]+(double)rebf[yc-r];
        }
        read(fp1, rebf, col+4);
        for (j=0; j<r+1; j++) {
            if (j==0) check=check+(double)rebf[yc];
            else check=check+(double)rebf[yc+j]+(double)rebf[yc-j];
        }
    }
jump: close(fp1);
    fp1=open(fileout, 0);
    if (check>=cpt) {
        cpt=arrck[1+n];
        rc[n]=r;
        n++;
    }
    if (check>=arrck[4]) break;
}

integer=row;
write(fp4, integer, 2);
integer=col;
write(fp4, integer, 2);
write(fp4, rc, 10);
/*
PROGRAM HOMO_PAS2 --- this routine enhances the input raster by employing
the homomorphic filtering technique.

compile with /usr/PPS/lib/psplib, /usr/PPS/lib/util.a
............................../lib/libm.a and /lib/libmath.a

chris'k. 'wu, 'Apr. '85
*/

#include "../usr/PPS/include/pps.h"
#include <stdio.h>
#include <math.h>
#define PMODE '0644
#define 'BUFSIZE '128

int row, col, rc[5];
char cmd[100];

main(ac, av)

int ac, char **av;
{
    int i, fp1, radius, c, flag;
    char filere[50], fileim[50], filee[50], filei[50], reout[50], imout[50];
    char rad[50];
    if (ac != 2) {
        printf("Usage: %s rastout\n", av[0]);
        printf("This routine enhances the input raster by employing\n");
        printf("homomorphic filtering technique\n");
        printf("rastout 'output raster filename\n");
        exit(0);
    }
    ppsinit(ac, av);
    flag=0;
    printf(filere, "readata");
    printf(fileim, "readata");
    printf(filee, "filtere");
    printf(filei, "filteri");
    printf(reout, "filteri");
    printf(imout, "filterc");
    printf(rad, "radius.h");
    fp1=open(rad, RD);
    read(fp1, &row, 2);
    read(fp1, &col, 2);
    read(fp1, &rc, 10);
    close(fp1);
```c
loop:
    if (!flag) {
        printf("new output filename ?\n");
        scanf("%s", av[11]);
        printf("%s\n", av[11]);
    }>

    printf("input radius as cutoff frequency\n");
    for (i=0; i<5; i++) printf("energy = %f at r = %d\n", 5.0/(float)i, rc[i]);
    printf("radius = \n");
    scanf("%d", &radius);
    printf("radius = %d\n", radius);
    filter(fslc, flc, lrc, lrc1, radius);
    sprintf(cmd, "jfft %s %s %s %d", fltc, flm, reout, iout, col);
    printf("%s\n", cmd);
    system(cmd);
    sprintf(cmd, "rm %s", iout);
    system(cmd);
    recover(reout, iout);
    sprintf(cmd, "plotift %s %s %d", iout, av[11], row, col);
    printf("%s\n", cmd);
    system(cmd);
    sprintf(cmd, "intdisp %s O O O %d %d %d %d\n", av[11], row, col);
    system(cmd);
    sprintf(cmd, "rm %s %s %s", flm, lrc, lrc1);
    system(cmd);
    error: "do you want to continue (y/n) ?"
        c=getchar();
        switch(c) {
            case 'y': flag=1,
            case 'n': break;
            default: goto 'error';
        }
    }
ppx_end();
exit();
}

power(z, n)
int z, n;
```
May '10 12:18 '1989  'homo_pas2.c  Page 3

```
<
int i, p;
p=1;
for (i=1; i<n; i++) p=p*x;
return(p);
>

filter(filere, fileim, fire, flim, radius)
int radius; 
char *filere, *fileim, *fire, *flim;
<

butworth(filere, fileim, fire, flim, radius);
high(filere, fileim, fire, flim);
>

/* BUTWORTH ----- butterworth 'filtering' subroutine */
butworth(redata, imdata, outre, outim, rad)
int 'rad;
char *redata, *imdata, *outre, *outim;
<
int i, j, fp1, fp2, fp3, fp4, xc, yc;
float *buf[BUSIZE], *buf2[BUSIZE], b, d;

fp1=fopen(redata, 0);
fopen(imdata, 0);
fp3=creat(outre, PMODE);
fp4=creat(outim, PMODE);
xc=row/2;
yc=col/2;
for (i=0; i<Crow; i++) 

```
.............. 'read(fp1, rbuf, col*4);
.............. 'read(fp2, ibuf, col*4);
.............. 'for (j=0; j<col; j++) 

```
................. 'd=(float)sqrt((double)((xc-i)*(xc-i)+(double)((yc-j)*(yc-j)));
................. 'b=1/(1 + d*d)' (float)(rad*rad));
................. 'rbuf[i][j]=rbuf[i][j]*b;

```
May '10 '12: 18 '1985: homo_pas2.c Page '4

...ibuf[j]=ibuf[j]*b;

...}

...write(fp3, rbuf, col#4);
...write(fp4, ibuf, col#4);

...}

close();

/>

/* 'HIGH' --- 'high emphasize filtering' */

high(rdata, idata, otre, ttim)
{

int i, j, fp1, fp2, fp3, fp4, fp5, fp6;
float *rbuf[BUFSIZE], *ibuf[BUFSIZE], *drbuf[BUFSIZE], *dibuf[BUFSIZE];
char *tp1[50], *tp2[50];

fp1=fopen(rdata, O);
fp2=fopen(idata, O);
fp3=fopen(otre, O);
fp4=fopen(ttim, O);
sprintf(fp1, "tpf11");
sprintf(fp2, "tpf12");
fp5_CREAT(tp1, 0, 0, 0);
fp6_CREAT(tp2, 0, 0, 0);
for (j=0; j<row; j++) {

.......
read(fp1, drbuf, col#4);
read(fp2, dibuf, col#4);
read(fp3, rbuf, col#4);
read(fp4, ibuf, col#4);
for (j=0; j<col; j++) {

.......
rrbuf[j]=1.95*drbuf[j]-rrbuf[j];
iibuf[j]=1.95*dibuf[j]-iibuf[j];

.......
}

.......
write(fp5, rbuf, col#4);
write(fp6, iibuf, col#4);

.......
}

close();
unlink(otre);
sprintf(cmd, "mv %s %s", tp1, otre);
printf("%s\n", cmd);}
system(cmd);
unlink(otim);
sprintf(cmd, "mv %s %s", tp2, otim);
printf("%s\n", cmd);
system(cmd);
}

recover(data1, data2)
char *data1, *data2;
{
int i, j, fp1, fp2;
float bufre[BUFSIZE];

fp1=open(data1, 0);
fp2=creat(data2, PMODE));

for (i=0; i<row; i++) {

    read(fp1, bufre, col#4);
    for (j=0; j<col; j++) { bufre[j]=(float)exp((double)bufre[j]);
    write(fp2, bufre, col#4);
    }
}
close();
}
/* PROGRAM SMOOTH -- SPATIAL AVERAGING ALGORITHM
CHRIS K. WU, DEC. 24, 1985*/
#include <stdio.h>
#include <math.h>
main(ac, av)
int ac, char *av[1];
{
 int outbuf[640];
 float te;
 int row,col,buf[7],i,j,k,l,rfp,wpf,err,*temp,*calloc();
 int dx,dy,wx,wy,outbuf[640],size,sum;
 if(ac!=5) {
   printf("usage: smooth filein fileout row col\n");
   printf("Spatial averaging algorithm\n");
   printf("filein -- input filename\n");
   printf("fileout -- output filename\n");
   printf("row,col -- file size\n");
   exit();
 }
 printf("window size (dx dy), the window size is determined by\n");
 printf("w=(2dx+1)*(2dy+1)n? ");
 scanf("%d %d",&dx,&dy);
 wx=2*dx+1;
 wy=2*dy+1;
 size=0;
 sum=0;
 row=atoi(av[3]);
 col=atoi(av[4]);
 for(i=0;i<wx;i++) buf[i]=calloc(col,1);
 rfp=fopen(av[2],"r");
 for (i=0;i<wx;i++) err=fopen(buf[i],2,col,wpf);
 for (i=0;i<dx;i++) err=fopen(buf[i],2,col,wpf);
 for (i=dx;i<row-dx;i++) {
   for (j=dy;j<dy;j++) outbuf[j]=buf[ds][i];
   for (j=dy;j<dy;j++) {
     for (k=0;k<wy;k++)
       for (l=j-dy;l<dy;l++) sum=sum+buf[l][k][i];
     outbuf[j]=sum/size;
     sum=0;
   }
   for (j=col-dy;j<col;j++) outbuf[j]=buf[ds][i];
   err=fopen(outbuf,2,col,wpf);
   if (i!=row-dx-1) {
     temp=buf[0];
     for (i=0;i<wl-1;i++) buf[i]=buf[i+1];
     buf[wl-1]=temp;
   }
   err=fopen(buf[wx-1],2,col,wpf);
   }
 for(i=dx+1;i<wx;i++) err=fopen(buf[i],2,col,wpf);
 fclose(rfp);
 fclose(wpf);
 exit();
}
#include "#/usr/PPS/include/pps.h"
#include <stdio.h>

main(ac, av)
int ac, char *av;
{
    char *file1[20], *file2[20], cmd[50];
    if(ac!=3)
        ........ print("usage: %s "file1 "fileout\n", av[0]);
        ........ print("program for raster smoothing with wiener filter\n");
        ........ print("filein -- input filename\n");
        ........ print("fileout -- output filename\n");
        ........ exit();
        ........
        if(!open(file1, "tspm1"));
        if(!open(file2, "tspm2"));
        system(cmd);
        system(cmd);
        printf(file2, "tspm2");
        printf(file2, "tspm2");
        system(cmd);
        system(cmd);
        system(cmd);
        pps_end();
        exit();
    }
/*PROGRAM 'WIENER_PASI' -- this routine separates the input raster
......................... into few regions according to the spatial
......................... activities

Chris H. Wu, Nov. '23, 1985
*/
#include "/usr/PPS/include/pps.h"
#include <stdio.h>
#include <math.h>

main(ac, av)
int ac;
char *av[];
{
    float buf[2562], fac, mean, min, max, var, region[5];
    int n, row, col, i, j, jy, fpl, fp[5], buf[5][512], bufout[5][512];
    char *file[5][20], cmd[100], file1[20];

    /*.......
     if(ac<3){
    repeat: printf("Usage: %s 'file1' 'file2' ... 'n' 'av[O]");
    printf("wiener_pas1 separates input files into few regions\n");
    printf("according to the spatial activities\n");
    printf("'filein' -- input raster 'filename'\n");
    printf("'n' -- # of regions\n");
    printf("'filei', 'file2', ... -- 'output filename'\n");
    exit();
    }
    ppsinit(ac, av);
    /*.......
    n=atoi(av[2]);
    printf("threshold 'scale(x) = mean+/-sigma', 'r"");
    scanf("%f", &fac);
    if '((n>3) && ac)' goto 'repeat;
sprintf(file1, "tfile1");
tps(av[1], file1, &min, &max, &mean, &var);
onopenfle((O, av[1], RD);
    row=H[0], nrow;
col=H[0], ncol;
    switch(n) {'
        case 2:
            region[0]=mean+fac*var;
            region[1]=max;
            break;
        case 3:
            region[0]=mean-(mean-min)/3;
            region[1]=mean+(max-mean)/3;
            region[2]=max;
            break;
        case 4:
            region[0]=mean-(mean-min)/2;
            region[1]=mean;
            region[2]=mean+(max-mean)/2;
            region[3]=max;
            break;
        case 5:
            break;
    } /*.......
*/
for(i=0; i<N; i++) {
    printf(file1[i], "tp_");
    strcat(file1[i], av[3+i]);
    fp[i]=creat(file1[i], 0644);
}

for(i=0; i<row1; i++) {
    readrow(fp[i], buf1);
    read(fp1, buf2, col*4);
    for(j=0; j<col; j++) {
        for(jj=0; jj<i; jj++) {
            if(buf2[jj] <= region[jj]) {
                bufout[j][jj]=buf1[jj];
                if(jj!=`(n-1))
                    while (jj+1 && bufout[jj+1][jj]=0;
            } else bufout[j][jj]=0;
        }
        for(jj=0; jj<i; jj++) write(fp[i][jj], bufout[j][jj], col*2);
    }
    close(fp1);
    unlink(file1);
    for(i=0; i<N; i++) {
        close(fp1);
        fp[i]=open(file1[i], 0);
        creat(file1[i], av[3+i], 0);
        for(j=0; j<row1; j++) {
            read(fp1, buf1, col*2);
            writerow(ii, bufii);
        }
        closefile1;
        closefp1;
        unlink(file1);
    }
}

ppx_end();
exit();

routine 'for spatial activities calculations'
spa(file1, file2, min, max, mean, var)
char *file1, *file2; *float *min, *max, *mean, *var;


toi row, col, *buf, *arralloc(), i, ii, j, x, fp1, *tp, power();
float *outbuf[512], ts;
double *temp1, temp2, sum0, sqsum0;

min=100;
max=0;
openfile(0, file1, RD);
readrow(0, buf[i1]);
for(i=0; i<5; i++)
   readrow(0, buf[i1]);
for(i=0; i<5; i++)
   for(j=0; j<col1; j++)
   {
      if(i==row1) {
         if(j==col1) {
            temp1=(double)power(buf2[i1][j]-buf[i1][j-1], 2);
            temp2=(double)power(buf2[i1][j]-buf[i1][j+1], 2);
            outbuf[j]=(float)sqrt(temp1+temp2);
         }
      }
      else {
         temp1=(double)power(buf2[i1][j+1]-buf[i1][j], 2);
         temp2=(double)power(buf2[i1][j+1]-buf[i1][j-1], 2);
         outbuf[j]=(float)sqrt(temp1+temp2);
      }
   }
else if(j==col1) {
   if(i==1) x=2;
   else x=1;
   temp1=(double)power(buf3[i1][j]-buf[i1][j-1], 2);
   temp2=(double)power(bufx[i1][j]-buf[i1][j+1], 2);
   outbuf[j]=(float)sqrt(temp1+temp2);
}
else if(i==0) 
   if(j==0) {
      if(i==1) x=2;
      else x=1;
      temp1=(double)power(buf3[i1][j-1]-buf[i1][j-2], 2);
      temp2=(double)power(bufx[i1][j-1]-buf[i1][j], 2);
      outbuf[j]=(float)sqrt(temp1+temp2);
   }
else {
   if(i==1) j=row2-1; 
   else if(i==0) j=col2-1;
      if(i=1) x=1;
   else x=2;
      temp1=(double)power(buf3[i1][j-1]-buf[i1][j-2], 2);
      temp2=(double)power(bufx[i1][j-1]-buf[i1][j-2], 2);
      outbuf[j]=(float)sqrt(temp1+temp2);
   }
else {
   if(outbuf[j]<min) min=outbuf[j];
   else if(outbuf[j]>max) max=outbuf[j];
   sum=sum+double(outbuf[j]);
   sqsum=sqsum+double(outbuf[j]*outbuf[j]);
   write(fp1, outbuf, col4);
```c

```
```
Nov 20 23:31 1985 'wiener_pas2.c Page 1

/*
PROGRAM WIENER_PAS2 -- this routine restores the input files
with the wiener filter.

compile with /usr/PPS/lib/ppslib, /usr/PPS/lib/util.a
chris k. wu, Nov. '85
*/

#include "<ppslib>"
#include "<stdio.h>
#include "<math.h>
#define 'P' 644

main(ac, av)

int 'ac', 'char '*='av';
{

float 'var', 'sqvar', 'd', 'd0', 'b', 'w', 'buf[5][512]', 'bufout[512]';
int 'i', 'j', 'k', 'n', 'row', 'col', 'xc', 'yc', 'fp[5]', 'fp1';
double ' atof()';
char *'filere[5][20]', *'fileim[5][20]', *'outre[20]', *'outim[20]', *'cmd[50]';
char *'outfile[5][20]', *'outfileim[5][20]';

if ('ac' < '5') {

repeat: print("Usage: %s 'fileout' 'filere' 'fileim' 'file2' ...
", av[0]);
        print("This routine restores the input file\n");
        print("by using the wiener filtering technique\n");
        print("fileout 'output' filename\n");
        print("n" *** '# of separated pictures\n");
        print("fileim, file2, ... picture filenames\n");
        exit();

        ppsinit(ac, av);
        n=atof(av[2]);
        if (n+3)='ac' goto repeat;
        openfilere(0, av[31], RN);
        row=n[0]. rows;
        col=n[0]. ncols;
        closefile(0);
        sprintf(outre, "tpre");
        sprintf(outim, "tpim");
        xc=row/2;
        yc=col/2;

        for ('i'=0; 'i'<n; 'i'++) {

            sprintf(filere[i], "re_");
            sprintf(fileim[i], "im_");
            strcat(fileim[i], "av3+i");
            strcat(filere[i], "av3+i");
            sprintf(cmd, "#ff %s %s %s", av[3+i], filere[i], fileim[i]);
            system(cmd);

        }
for (k=0; k<n; k++) {
    for (i=0; i<n; i++) {
        d0=dist(float) exp(-0.693147*(double)(n-k)/(double)(n));
        mean_var=(av[i+k&3]+&var);
        sqvar=2*var*var;
        for (j=0; j<4; j++) {
            read(fp[i], buf[i], col=4);
            for (j=0; j<4; j++) {
                d=(float) sqrt((double)((xc-i)*(xc-i)+(yc-j)*(yc-j)));
                b=(b*b+sqvar)/d0;
                buf[i][j]=buf[i][j]*b;
            }
        }  
        write(fp[i], buf[i], col=4);
    }
}
for (i=0; i<4; i++) close(fp[i]);
unlink(filein[k]);
unlink(fileout[k]);
sprintf(cmd, "mv %s %s", out, filein[k]);
system(cmd);
sprintf(cmd, "mv %s %s", out, fileout[k]);
system(cmd);
sprintf(outfile, "tempout");
for (i=0; i<n; i++) {
    sprintf(outfile[i], out_"");
    strcat(outfile[i], av[k+3]);
    sprintf(cmd, "ifpt %s %s %s %d", filein[i], outfile[i], outfile[i], outfile[i], system(cmd);
    sprintf(cmd, "rm %s %s", filein[i], outfile[i]);
system(cmd);
}
for (i=0; i<n; i++) {fp[i]=open(outfile[i], O_RDONLY);
fp[i]=creat(outfile[i], PMODE);
for (j=0; j<n; j++) {
    read(fp[j], buf[j], col=4);
    for (j=0; j<n; j++) {
        bufout[j]=0;
    }
    for (k=0; k<n; k++) {bufout[j]=bufout[j]+buf[k][j];
    }
    write(fp, bufout, col=4);
    }
}
unlink(outfile[i]);
sprintf(cmd, "plotfit %s %s %d", outfile[i], row, col);
system(cmd);
unlink(outfile);
pps_end();
exits();
>
mean_var(#file,#var)
char *file; float *var;
{
    float mean=0, var1=0;
    int i, j, row, col, buf[512], count=0;
    openfile(0, file1, RD);
    row=h[0].nrows;
    col=h[0].ncols;
    for(i=0; i<row; i++) {
        for(j=0; j<col; j++) {
            if(buf[j]!=0) {
                mean+=buf[j];
                var1+=buf[j]*buf[j];
                count++;
            }
        }
    }
    mean/=count;
    *var=(float)sqrt((double)((var1/count-mean*mean)));
    closefile(0);
}
/ * PROGRAM FD -- THIS PROGRAM DETECTS EDGES BY USING THE FIRST 
DERIVATIVE EDGE DETECTION METHOD WITH THRESHOLD 

CHRIS K. VU, NOV. 26, 1985 */ 
#include <stdio.h> 
#include <math.h> 
main(ac, aw) 
int ac, char *av[1]; 
{ 
char file[20], ans; 
int row, col, *buf[2], *tep, i, j, rfp, wfp, err, *alloc(); 
float te, outbuf[640], var, mean, count; 
double temp1, temp2, thre, sum, sqsum; 
snum=0.; 
sqsum=0.; 
count=0.; 
if(ac!=5) 
{ 
print("usage: fd filein fileout row col\0n"); 
print("First derivative edge detector\0n"); 
print("filein -- input filename\0n"); 
print("fileout -- output filename\0n"); 
print("row, col -- file size\0n"); 
exit(); 
} 
sprintf(file1, "tempfd"); 
row=atoi(av[3]); 
col=atoi(av[4]); 
for(i=0; i<2; i++).buf[i]=alloc(col, 2); 
repeat: 
printf("had the gradients been calculated (y/n) ? "); 
scanf("%c", &ans); 
if(ans=='y') goto jump; 
if(ans=='n') goto cont; 
else goto repeat; 
cont: 
rfp=fopen(av[13], "r"); 
wfp=fopen(file1, "w"); 
for(i=0; i<2; i++) 
e= fread(buf[i]+2, col, rfp); 
for(i=0; i<row; i++) 
for(j=0; j<col; j++) 
if(i==row-1) /* last row */ 
if(j==col-1) 
{ 

temp1=fabs(pow(double)(buf[i][j]-buf[i][j-1]), 2.)); 

temp2=fabs(pow(double)(buf[i][j]-buf[i][j+1]), 2.)); 

outbuf[i][j]=(float)sqrt(temp1+temp2); 
} 
else 
{ 

temp1=fabs(pow(double)(buf[i][j+1]-buf[i][j]), 2.)); 

temp2=fabs(pow(double)(buf[i][j]-buf[i][j+1]), 2.)); 

outbuf[i][j]=(float)sqrt(temp1+temp2); 
} 
else if(j==col-1) /* last column */ 
{ 

temp1=fabs(pow(double)(buf[0][j]-buf[0][j-1]), 2.)); 

temp2=fabs(pow(double)(buf[0][j]-buf[0][j+1]), 2.)); 

outbuf[i][j]=(float)sqrt(temp1+temp2); 
} 
else 
{ 

temp1=fabs(pow(double)(buf[0][j+1]-buf[0][j]), 2.)); 

temp2=fabs(pow(double)(buf[0][j]-buf[0][j+1]), 2.)); 

outbuf[i][j]=(float)sqrt(temp1+temp2); 
}
temp2=fabs(pow((double)(buf[i][j]-buf[0][j]),2));
outbuf[j]=((float)sqrt(temp1+temp2));
}
if (outbuf[j]<1.) {
    count=count+1.;
    sum=sum+(double)outbuf[j];
    sqsum=sqsum+(double)(outbuf[j]*outbuf[j]);
}
}
err=fwrite(outbuf,4,col,wfp);
if (i<row-2) {
    tep=buf[0];
    buf[0]=buf[1];
    buf[1]=tep;
    err=fread(buf[1],2,col,rfp);
}
fclose(rfp);
fclose(wfp);
mean=(float)(sum/(double)count);
sqsum=sqsum/(double)count-(double)(mean*mean);
var=(float)sqrt(sqsum);
printf("mean=%f, var=%f \n",mean,var);
jump:
printf("threshold scale(x), T=mean+x*var, ? ");
scanf("%f", &te);
thes=(double)te;
thes=mean+thes*var;
rfp=fopen(file1,"r");
wfp=fopen(sv21,"w");
for(i=0;i<row;i++) {
    err=fread(outbuf,4,col,rfp);
    for(j=0;j<col;j++) {
        if(outbuf[j]<thes) buf[0][j]=0;
        else buf[0][j]=63;
    }
    err=fwrite(buf[0],2,col,wfp);
}
fclose(rfp);
fclose(wfp);
exit();
/* PROGRAM SOBEL -- THIS PROGRAM DETECTS EDGES BY USING THE SOBEL EDGE DETECTOR */

chris k. wu, nov. 26, 1985

#include <stdio.h>
#include <math.h>
main(ac, av)
int ac, char *av[1];
{
    char filename[20], ans;
    float te, outbuf[640], mean, var, fac, thres;
    double temp1, temp2, temp3, g1, g2, g3, g4, sum, sqsum;
    int count=0, row, col; *buf[3], i, i1, j, x, rfp, wp, err, *temp, *calloc();
    if(ac!=2) {
        printf("usage: sobel filename outfilename row col\n");
        printf("Sobel edge detector\n");
        printf("filename -- input filename\n");
        printf("outfilename -- output filename\n");
        printf("row, col -- file size\n");
        exit();
    }
    printf(file, "tfill\n");
    row=atoi(av[3]);
    col=atoi(av[4]);
    for(i=0; i<3; i++) buf[i]=calloc(col, 2);
    repeat:
        printf("Have the gradients been calculated (y/n) ?\n");
        scanf("%c", &ans);
        if (ans==\'y\') goto jump;
        if (ans==\'n\') goto cont;
        else goto repeat;
    cont:
    sum=0;
    sqsum=0;
    rfp=fopen(av[1], "r");
    wp=fopen(file, "w");
    for(i=0; i<3; i++) err=fopen(buf[i], 2, col, rfp);
    for(i=0; i<row; i++) {
        for (j=0; j<col; j++) {
            if (i==row-1) { /* last row */
                temp1=fabs(pow((double) (buf[i][j]-buf[i][j-1]), 2));
                temp2=fabs(pow((double) (buf[i][j]-buf[i][j]), 2));
                temp3=sqrt(temp1+temp2);
                outbuf[j]=(float)temp3;
            } else {
                temp1=fabs(pow((double) (buf[i][j+1]-buf[i][j]), 2));
                temp2=fabs(pow((double) (buf[i][j]-buf[i][j]), 2));
                temp3=sqrt(temp1+temp2);
                outbuf[j]=(float)temp3;
            }
        }
    }
    else if(j==col-1) { /* last column */
        if (j<row-1) {
            temp1=fabs(pow((double) (buf[i][j+1]-buf[i][j]), 2));
            temp2=fabs(pow((double) (buf[i][j]-buf[i][j]), 2));
            temp3=sqrt(temp1+temp2);
            outbuf[j]=(float)temp3;
        }
    }
}


else if(i==0 || j==0) /* first row of column */
    if(i>0) x=1;
  else x=0;
  temp1=fabs(pow((double)(buf[i][j+1]-buf[i][j]),2.),2.);
  temp2=fabs(pow((double)(buf[i+1][j]-buf[i][j]),2.),2.);
  temp3=sqrt(temp1+temp2);
  outbuf[j]=(float)temp3;
}
else {
  temp1=(double)(buf[0][j-1]+buf[0][j+1]-buf[1][j-1]+buf[2][j+1]);
  temp2=(double)(buf[0][j]-buf[2][j]);
  g1=fabs(pow((temp1+1.414*temp2),2.,2.));
  temp1=(double)(buf[0][j-1]-buf[0][j+1]+buf[1][j-1]-buf[2][j+1]);
  temp2=(double)(buf[1][j]-buf[1][j+1]);
  g2=fabs(pow((temp1+1.414*temp2),2.,2.));
  temp1=(double)(buf[1][j-1]+buf[2][j]-buf[0][j]-buf[2][j+1]);
  temp2=(double)(buf[0][j+1]-buf[2][j-1]);
  g3=fabs(pow((temp1+1.414*temp2),2.,2.));
  temp1=(double)(buf[1][j-1]-buf[2][j]-buf[0][j+1] buf[1][j+1]);
  temp2=(double)(buf[0][j+1]-buf[2][j-1]);
  g4=fabs(pow((temp1+1.414*temp2),2.,2.));
  temp3=sqrt(g1+g2+g3+g4);
  outbuf[j]=(float)temp3;
}
if(outbuf[j]>1.) {
  count=count+1;
  sum+=sum+(double)outbuf[j];
  sqsum+=sqsum+(double)(outbuf[j]*outbuf[j]);
}
}
error=write(outbuf,4,col,wfp);
if (i>0) {
  temp=buf[0];
  for(i=0; i<(2*row); i++) buf[i]=buf[i+1];
  buf[2]=temp;
  if (i<row-2) error=fread(buf[2],2,col,rfp);
}
mean=(float)(sum/(double)count);
temp1=sqsum/(double)count;
var=(float)sqrt(temp1-(double)(mean*mean));
printf("mean =%.6f, variance=%.6f\n",mean,var);
jump:
printf("threshold factor (x), T=mean+x*variance, ? ");
scanf("%d", &fac);
thr=mean+fac*var;
close(wfp);
close(rfp);
rfp=fopen(file,"r");
wfp=fopen(sx[21],"w");
for(i=0; i<row; i++) {
  error=fread(outbuf,4,col,rfp);
  for(j=0; j<col; j++) {
    if(outbuf[j]>thr) buf[0][j]=0;
    else buf[0][j]=63;
  }
  error=fwrite(buf[0],2,col,wfp);
}
close(rfp);
fclose(wfp);
exit();
}
/*
  PROGRAM TEMPLATE — this program decomposes the input picture into
  a summation of basis subpictures using template
  matching technique.

  compile with `/usr/PSS/lib/ppslib` and `/usr/PSS/lib/util.a
  Chris K. Wu, Apr. '85
*/

#include "usr/PSS/include/pps.h"
#include <stdio.h>
#include <math.h>
#define ROWS 3

int **row, x[93],
float *tp[9193] = {
   ..., (1, 1.414, 1.1, 0.1, 0.1, -1.1, -1.414, -1.1, ...),
   ..., (1, 0.1, 1.414, 0.1, -1.414, 1.1, 0.1, -1.1, ...),
   ..., (0.1, -1.1, 1.1, 0.1, -1.414, 1.1, 0.1, -1.1, ...),
   ..., (1.1, -1.1, 1.1, 0.1, -1.414, 1.1, 0.1, -1.1, ...),
   ..., (1, 1.1, -2.1, 1.1, 1.1, -2.1, 1.1, 1.1, 1.1),
   ...,
};

float at[9] = {2, 2824, 2, 8284, 2, 8284, 2, 1, 0, 0, 0};

main(argc, argv)
int argc; char **argv;
{
int reply, i, ii, rows, ans, j;
int **arralloc(), **outbuf, COLS, *talloc(), *temp;
float *scale;
char 'mod1003';

if(argc != 4) {
   printf("Usage: %s filein fileout1 fileout2 \n", argv[0]);
   printf("this routine detects the features in the picture\n");
   printf("by using template matching technique\n");
   printf("filein — the input PPS raster \n");
   printf("fileout1 — the output raster of projection onto\n");
   printf("edge subspace\n");
   printf("fileout2 — the output raster of projection onto\n");
   printf("line subspace\n");
   exit();
}

printf("intensity contrast .1(very low) — .1(very high) (.1-.1)\n");
scanf("%.1f", &scale);
for(i=0; i<9; i++) tp[i+1] = tp[i+1] + scale;
ppsinit(argc, argv);
openfile(0, argv[1], RD);
ppsreport(0);

COLS=h[0].ncols;
rows=h[0].nrows;

outbuf=arralloc(2, COLS);
row=arralloc(ROWS, COLS);

createfile(1, argv[2], 0);
createfile(2, argv[3], 0);
ancestor(2, 0);
ancestor(1, 0);
sprintf(mod, "segmented by 'template' matching 'tech.' , 'projection' onto 'edge' subspace. ")
strcat(h[1].header, mod);
hflush(1);
sprintf(mod, "segmented by 'template' matching 'tech.' , 'projection' onto 'line' subspace. ")
strcat(h[2].header, mod);
hflush(2);
findrow(1, 0);
findrow(2, 0);

/*Initialize rows*/
for(i=0; i<ROWS; i++)
  if (((reply=readrow(0, row[i]))!=COLS+2) <
      fprintf(stderr, "read error when initialize rows\n");
      exit();
      >

if (((reply=writerow(1, outbuf[0]))!=COLS+2) <
    fprintf(stderr, "write error at first row\n");
    exit();
    >

writerow(2, outbuf[1]);

for (i1=1; i1<rows-1; i1++) {
  for (i=1; i<COLS-1; i++) {
    for (j=0; j<3; j++) {
    
    k[i,j]=row[0][i-1+j];
    t[i,j+3]=row[1][i-1+j];
    t[i+j+6]=row[2][i-1+j];
    }

    ans=comp();
    switch(ans) {
      case 0:
    
    outbuf[0][i]=0;
  
"}
Apr 23 22:43 1986 ~template.c Page 3

outbuf[1][i]=0;
break;
case 1:
outbuf[0][i]=h[0].ceiling;
outbuf[1][i]=0;
break;
case 2:
outbuf[0][i]=0;
outbuf[1][i]=h[0].ceiling;
break;
case 3:
outbuf[0][i]=h[0].ceiling;
outbuf[1][i]=h[0].ceiling;
break;
}

if ((reply=writerow(1, outbuf[0])) != 'COLS#2') {
    fprintf(stderr, "write 'error' on row %d, %d bytes written\n", i, reply);
exit();
}

writerow(2, outbuf[1]);
if (i<rows-2) {
    temp=row[0];
    for (i=0; i<2; i++) row[i]=row[i+1];
    row[2]=temp;
    if ((reply=readrow(0, row[2])) != 'COLS#2') {
        fprintf(stderr, "read 'error' on row %d\n", i);
exit();
}
}

for (i=0; i<COLS; i++) row[0][i]=0;
writerow(2, row[0]);
writerow(1, row[0]);
pps_end();
exit();
}

comp()
{
float t, t1, t2, p, p1, p2, s;

int ans, i, j;

t=0.1;
tl=0.1;

tn=0.;
pe=0.;
pl=0.;
pn=0.;

for (i=0; i<4; i++) {
    for (j=0; j<4; j++) {
        te=te+tp[i][j]*(float)x[j];
        tl=tl+tp[i+4][j]*(float)x[j];
    }
    te=te/atl[i];
    tl=tl/atl[i+4];
    pe=pe+(te*te);
    pl=pl+(tl*tl);
    te=0.;
    tl=0.;
}

for (i=0; i<4; i++) {tn=tn+tp[8][i]*(float)x[i];
    tn=tn/at[8];
    pn=tn*tn;
    printf("%f, %f, %f\n", pn, pe, pl);
    ax=(float)sqrt((double)(pe+pl+pn));
    if (ax==0.) return(0); 
    pl=(float)sqrt((double)pe);
    pn=(float)sqrt((double)pn);
    thetan=(float)fabs(acos((double)(pn/ax)));
    thetac=(float)fabs(acos((double)(pe/ax)));
    thetal=(float)fabs(acos((double)(pl/ax)));
}/

printf("%f, %f, %f\n", thetan, thetac, thetal);
if ((thetac>thetan) && (thetal>thetan)) ans=0;
else if (thetac==thetal) ans=3;
else if (thetac<thetal) ans=2;
else ans=1;

return(ans);
/*PROGRAM SEGMENT--This program divides the input picture into segments
   by using the histogram segmentation method.

Chris K. Wu, Jan. 27, 1985
*/
#include <stdio.h>
#include <math.h>
main(int argc, char *argv[])
{
    int row, col, n, i, j, seg[10], table[10];
    int rfp, wfp, err, k, buff[640], div;
    if (argc != 5) {
        printf("usage: segment filein fileout row col \n");
        printf("this program divides input picture into segments \n");
        printf("by using the histogram segmentation method\n");
        printf("filein -- input data filename\n");
        printf("fileout -- output data filename\n");
        printf("row, col -- file size\n");
        exit(0);
    }
    rfp = fopen(argv[1], "r");
    wfp = fopen(argv[2], "w");
    row = stoi(argv[3]);
    col = stoi(argv[4]);
    printf("input # of segments ? \n");
    printf("input limits in the order from low to high\n");
    scanf("%d", &n);
    for(i=0; i<n; i++) {
        printf("upper limit for segment # %d ? \n", i+1);
        printf("\n");
        scanf("%d", &seg[i]);
    }
    div = 63/n;
    for(i=0; i<n; i++) {
        if((div * (i+1)) % 63) table[i] = 63;
        else table[i] = div * (i+1);
    }
    for(i=0; i<row; i++) {
        err = fread(buf, 2, col, rfp);
        for(j=0; j<col; j++) {
            for(k=0; k<63; k++) {
                if(buf[k] == seg[k]) {
                    buff[k] = table[k];
                    k = 0;
                }
            }
        }
        err = fwrite(buf, 2, col, wfp);
    }
    fclose(rfp);
    fclose(wfp);
    exit(0);
}
/* PROGRAM SIZE- This program measures the grain sizes of the input
wireframe picture by using the label-merge algorithm

CHRIS K. WU, FEB. 19, 1986 */
#include <stdio.h>
#include <math.h>
int result[][6000],result1[6000];
double sum=0, btotal=0, sqarea=0;
main(ac, ay)
int ac, char *av[3];
{
int row, col, i, j, rfp, wfp, err, *oldline, *newline, *calloc();
int grain[200], edge[200], *cgrain, *clength;
int *oedge, *oleft, *ocount=0;
int ilength=0, label, count=0, isize, combine, bgrain=0, ingrain=0;
int jf, k, flag, flag2, temp;
double scale, tarea, total, st=0;
float xi, yi;
if(ac!=5) message();
row=atoi(av[3]);
col=atoi(av[4]);
cegrain=calloc(200, 2);
clength=calloc(200, 2);
oedge=calloc(200, 2);
oleft=calloc(col, 2);
newline=calloc(col, 2);
printf("Input the actual dimension of the full screen picture\n");
printf("in mm (x,y) ?\n");
scanf("%f%f", &xi, &yi);
tarea=(double)x1=(double)y1;
scale=tarea/(600.*600.);
for (i=0; i<200; i++) {
edge[i]=0;
grain[i]=0;
}
for (i=0; i<6000; i++) result[i][j]=0;

/****************************
area counting algorithm
******************************/
rfp=fopen(av[1], "r");
wfp=fopen(av[2], "w");
/****************************
label the grains in first row
and remember these grains as
the boundary grains
***************************************************************************/
e= fread(oldline, 2, col, rfp);
label=1;
for (j=0; j<col; j++) {
if (oldline[j]==43) { 
  oldline[j]=label;
  length++;
} else if (oldline[j]==63 && length!=0) { 
  edge[0]=label;
  grain[0]=length+2;
  ograin[0]=label;
  olength[0]=length;
  }
oaccount++;  
label++;  
length=0;  
}  
if (j==col-1 & length!=0) {  
  edgelabel=length+1;  
grain(label)=length+1;  
ograin(oaccount)=label;  
olength(oaccount)=length;  
count++;  
length=0;  
}  
errfwrite(oldline,2,col,wpf);  
/****************************  
label the rest of grains in picture  
*******************************/  
for (i=1;i<row;i++) {  
  errfread(newline,2,col,rfp);  
  /****************************  
label the grains in each row  
*******************************/  
for (j=0;j<col;j++) {  
  if (newline[j]==63) {  
    /*************************************************************  
    check if this segment belongs to a labelled grain  
    **********************************************/  
    if (oldline[j]!=0) {  
      label=oldline[j];  
      /*********************************  
      backward labelling  
      *******************************/  
      if (j>0) {  
        jj=j-1;  
        while (newline[jj]==63 & jj!=0) {  
          /*******************************  
          remember if it is a boundary grain  
          *******************************/  
          if (jj==0) edgelabel(label,edge);  
          newline[jj]=label;  
          length++;  
          jj--;  
        }  
      }  
    }  
  }  
  /******************************  
  forward labelling  
  ******************************/  
  while (newline[j]==63 & j<col) {  
    /*******************************  
    remember if it is a boundary grain  
    *******************************/  
    if (j==col-1) edgelabel(label,edge);  
    newline[j]=label;  
    length++;  
    /*******************************  
    check if two grains have to be combined  
    *******************************/  
    if (oldline[j]=label & oldline[j]!=0) {  
      combine=oldline[j];  
      grainlabel=(grainlabel)+grain(combine);  
    }  
  }  
}
grain(combine)=0;
}
}
j++;
}
j--;

/**********************************************************************
check if a new grain is encountered
***************************************************************************/
else if (j+1<col) {
  if (newline[j+1][13]==63) {
    /**********************************************************************/
    find a new label for this grain
    **********************************************************************/
    for (k=1;k<200;k++) {
      if (grain(k)==0) {
        label=k;
        k=200;
      }
    }
    /**********************************************************************/
    Backward labelling
    *************************/
    jj=j;
    while (newline[jj][13]==63 && jj>0) {
      if (jj==0) edgeck(label,edge);
      newline[jj]=label;
      length++;
      jj--;
    }
    length=length+2;
  }
}

/*******************************************************************************
check if the last column is encountered
*******************************************************************************/
if (j==col-1) {
  /**********************************************************************/
  add the length of the segment to the grain to which it belongs
  **********************************************************************/
  if (length>0) {
    grain[label]=grain[label]+length+1;
    length[count]=length;
    length=0;
    grain[count]=label;
    count++;
  }
}

/***********************************************************************
else a new grain is encountered
***************************************************************************/
else {
  for (k=1;k<200;k++) {
    if (grain(k)==0) {
      label=k;
      k=200;
    }
  }
  edgeck(label,edge);
  jj=j;
while (newline[jj]==63 && jj==0) {
    newline[jj]=label;
    length++;
    jj--;
}
length=length*2;
}

/******************************/
add the length to the grain it belongs to if
end of the segment is reached
******************************/
else if (newline[jj]==63 && length>0) {
    grain[label]=grain[label]+length+2;
    length=0;
    cgrain[count]=label;
    count++;
}

/******************************/
add boundary pixels to corresponding grains
******************************/
for (j=0;j<count;j++) {
    for (k=0;k<count;k++) {
        if (cgrain[j]==cgrain[k]) {
            diff=(int)fabs((double)length[j1-(double)length[k1]);
            if (diff<2) {
                label=cgrain[j];
                grain[label]=grain[label]+diff-2;
            }
            k=count;
        }
    }
}

/******************************/
remember the area of a grain if it is completely counted
******************************/
for (j=1;j<200;j++) {
    if (grain[j]==0) {
        /******************************/
        check if a grain is completely counted
        ******************************/
        flag=0;
        for (k=0;k<count;k++) {
            if (j==cgrain[k]) {
                flag=1;
                k=count;
            }
        }
        if (flag==1) {
            /******************************/
            delete this grain if it is less than 100 pixels
            ******************************/
            if (grain[j]<100) {
                st=st+1;
                grain[j]=0;
                for (k=0;k<200;k++)
                    if (edge[k]==j) edge[k]=0;
check if this grain is a boundary grain

else {
    for (k=0;k<200;k++) {
        if (edge[k] == 0) {
            if (j == edge[k]) {
                btotal = btotal + (double) grain[j];
                edgen[k] = 0;
                grain[j] = 0;
                flag = 2;
                bgrain++;
        
        }
    }

    remember the size of this grain if it is not a boundary grain

    if (flag == 2) {
        size = grain[j];
        sum = sum + (double) size;
        result[size]++;
        ingrain++;
        grain[j] = 0;
    }
}

}

write(newline, 2, col, wfp);
ncount = count;
temp = cgrain;
ograin = cgrain;
cgrain = temp;
temp = olength;
olength = clength;
clength = temp;
temp = newline;
newline = oldline;
oldline = temp;
if (i * row - 1) count = 0;

count the number of the incomplete boundary grains

bgrain = bgrain + count;
total = (double) row * (double) col;
compensate(st);
output(total, scale, ingrain, bgrain);
exit();

subroutine for compensation of ignored grains

compensate(st)

double st;
{  
  int i,j,compset;
  double diff,tcompsat=0.;
  diff=btOTAL+SUM;
  for(i=1;i<6000;i++)  
    
      if (result[i]!=0)  
        compsat=(int)(st*(double)i/diff);
      result[i+compset]=result[i];
  for(j=1;j<result[i]+1;j++)  
    square=square+(double)(i+compset)*(double)(i+compset);
  tcompsat=tcompsat+(double)compsat;
  
}  

SUM=sum+tcompsat;
BTOTAL=BTOTAL+St-Tcompsat;

//*********************************  
program output subroutine  
**********************************  
output(total,scale,ingrain,bgrain)  
int ingrain,bgrain;
double total,scale;
{
  int OFP,i;
double mean, var;
  ofp=fopen("1st","w");
  fprintf(ofp,"The area of processed picture:\n");
  fprintf(ofp,"%d %2\n\n",total*scale);
  fprintf(ofp,"The grains of size less than \n");
  fprintf(ofp,"%d %2 (%10.5f) are ignored but in", scale*100.);
  fprintf(ofp,"the loss is compensated into each output grain\n");
  fprintf(ofp,"Total number of grains : %d\n",ingrain+bgrain);
  fprintf(ofp,"Boundary grains : %d\n",bgrain);
  fprintf(ofp,"Boundary grains : %d\n",btotal*scale,btotal/total);
  fprintf(ofp,"Inner grains : %d\n",ingrain);
  fprintf(ofp,"%d %2 (%10.5f)\n",sum*scale,sum/total);
  fprintf(ofp,"The grain size distribution :\n");
  fprintf(ofp,"area grains\n");
  fprintf(ofp,"area grains\n");
  for (i=1;i<6000;i++)  
    
      if (result[i]!=0)  
        fprintf(ofp,"%8.3e %3d \n",(double)i*scale,result[i]);
    
}  

mean=SUM/(double)INGRAIN;
var=sqrt(square/(double)INGRAIN)-mean*mean;
fprintf(ofp,"The average area per grain :\n");
fprintf(ofp,"%8.3e %2\n\n",mean*scale);
fprintf(ofp,"The standard deviation of the distribution curve :\n");
fprintf(ofp,"%8.3e %2\n\n",var*scale);
fprintf(ofp,"The average grains per mm\n\n");
fprintf(ofp,"%f grains\n",1./(mean*scale));

//*********************************  
subroutine for checking a boundary grain  
against a list of boundary grains  
**********************************  
edgecheck(label,edge)
int label,edge[200];
{
  int I,flag=0;
}
/*
  return if it is in the list
  *****************************************
  for (i=0;i<200;i++) {
    if (edge[i]==0) {
      if (edge[i]==label) {flag=1; i=200;}
    }
  }

  otherwise add it into the boundary list
  *****************************************
  if (flag==1) {
    for (i=0;i<200;i++) {
      if (edge[i]==0) {
        edge[i]=label; i=200;}
    }
  }

/*
  message for program usage
  *****************************************/
  message() {
    printf("usage: size filein fileout row col
");
    printf("Size distribution curve generator
");
    printf("filein -- filename of input edge picture\n");
    printf("fileout -- output filename which contains the \n");
    printf("    labelled grain picture\n");
    printf("row,col -- file size\n");
    exit(0);}
}
/*
PROGRAM FFT -- This program calculates the 2-D Fourier Transform
               of the input raster. The origin is shifted to
               the center of the output raster.
compile with /usr/PPS/lib/ppslib /usr/PPS/lib/utile.a
               /lib/libm.a and /lib/libmath.a
chris k. wu, Apr. '86
*/

#include "'/usr/PPS/include/pps.h"
#include <stdio.h>
define PMODE 0644

main(argc,argv)
int argc; char *argv;\n{

char *cmd[250], temp1[250], temp2[250];
int reply, i, j, bufsize, n, m, nt, rows, cols, fp1, fp2, fp3, fp4;
int *temp[128];
double sum=0.0;
float rebuf[128], imbuf[128];

if(argc != 4) '{
   printf("Usage: %s <filein >reda >imdata >n", argv[0]);
   printf("this routine calculates the 2-D FT of the input\n");
   printf("raster. \n\n");
   printf("Input file size is restricted at maximum 128x128\n");
   printf("filein \> input PPS raster >filename\n");
   printf("redata \> real data output >filename\n");
   printf("imdata \> imaginary data output >filename\n");
   exit();
}'

ppsinrt(argc, argv);
openfile(0, argv[11], RD);

fp1=create(argc[21], PMODE);
fp2=create(argc[33], PMODE);

if((rows=argv[0], n) > 128 || (cols=argv[1], ncols) > 128) '{
   printf("Input file size is restricted at maximum 128x128\n");
   exit();
}'
bufsize=cols*2;

/* Perform FT of each row */

nt=cols;
n=cols;
m=0;
while nt>1 ){
Apr '23 22:46 1986 "fft.c Page 2"

```c

for (i=0; i<nrows; i++) {

    if (reply = readrow(0, temp)) != 'bufsize') {

        fprintf(stderr, "read error on row %d, %d bytes read\n", i, reply);
        exit();
    }

    for (j=0; j<cols; j++) {

        nt += 1;
        imbuf[j]*=1;
        rebuf[j] = (float) power(-1, nt)*(float) temp[j];
    }

    fft2(rebuf, imbuf, n, m, 1);
    write(fp1, rebuf, bufsize*2);
    write(fp2, imbuf, bufsize*2);
}

close(fp1);
close(fp2);
sprintf(cmd, "flipreal %s %d", argv[2], cols);
system(cmd);
sprintf(cmd, "flipreal %s %d", argv[3], cols);
system(cmd);

/* Perform FT of each column */

n = rows;
n = n;
m = 0;
while (m < 1) {
    nt = nt/2;
    m++;
}

fp1 = open(argv[2], O);
fp2 = open(argv[3], O);
sprintf(temp1, "tempr");
sprintf(temp2, "tempi");
fp3 = creat(temp1, PMODE);
fp4 = creat(temp2, PMODE);

for (i=0; i<cols; i++) {

    read(fp1, rebuf, bufsize*2);
    read(fp2, imbuf, bufsize*2);
    fft2(rebuf, imbuf, n, m, 1);
```
Apr '23 22:46 '96 5 ft. c Page 3

```
write(fp3, rebuf, bufsize=2);
write(fp4, imbuf, bufsize=2);

close(fp1);
close(fp2);
close(fp3);
close(fp4);
unlink(argv[2]);
sprintf(cmd, "mv 'temp r %s', argv[2]");
system(cmd);
unlink(argv[3]);
sprintf(cmd, "mv 'temp i %s', argv[3]");
system(cmd);

sprintf(cmd, "flipreal %s %d", argv[2], cols);
system(cmd);
sprintf(cmd, "flipreal %s %d", argv[3], cols);
system(cmd);

pps_end();
exit();
```

```c
power(x, n)
int x, n;
{
    int i, p;
p=1;
if (n==0) return(1);
for (i=1; i<n; i++) p=p*x;
return(p);
}
```
SUBROUTINE 'FLIPREAL' -- This routine 'flips' the input 'floating point
raster' along the 'diagonal' axis for the 2-D FFT program.

Compile with 'util.a' and 'ppslib
written by Chris K. Wu, Apr. 1, 1985
*/

#include "/usr/PPS/include/pps.h"
define PMODE '0644

main(ac, av)
int ac, *char **av;
{
char *tmpfile[50], cmd[256];
int SIZE;

if(ac!=3) {
"helpmsg(av[0]);
""exit(1);
""
ppslinit(ac, av);
SIZE=stoi(av[2])/2;
sprintf(tmpfile, "tempfile");
dorotate(av[1], tmpfile, SIZE);
unlink(av[1]);
sprintf(cmd, "mv 'tempfile %s', av[1]);
system(cmd);
pps_end();
exit(0);
}

/*
DOROTATE() -- Handle rotations of the 'raster' file.
*/
dorotate(raster, tmpfile, SIZE)
int SIZE, *char *raster,*tmpfile;
{
float *buf1[64][64], *buf2[64][64], temp[128], dt;
int i, j, fp, fp1;
fp=open(raster, 0);
for (i=0; i<SIZE; i++){

for (j=0; j<
`read(fp, buf1[i], SIZE=4);`  
`read(fp, buf2[i], SIZE=4);`  

```c
close(fp);
fp=open(raster, 2);
for (i=0; i<SIZE; i++) {
    for (j=i+1; j<SIZE; j++) {
        dt=buf1[i][j];
        buf1[i][j]=buf1[j][i];
        buf1[j][i]=dt;
    }
}
for (i=0; i<SIZE; i++) {
    write(fp, buf1[i], SIZE=4);
    write(fp, buf2[i], SIZE=4);
}
for (i=0; i<SIZE; i++) {
    read(fp, buf1[i], SIZE=4);
    read(fp, buf2[i], SIZE=4);
}
close(fp);
fp=open(raster, 2);
for (i=0; i<SIZE; i++) {
    read(fp, temp, SIZE=8);
    for (j=i+1; j<SIZE; j++) {
        dt=buf2[i][j];
        buf2[i][j]=buf2[j][i];
        buf2[j][i]=dt;
    }
}
for (i=0; i<SIZE; i++) {
    write(fp, buf1[i], SIZE=4);
    write(fp, buf2[i], SIZE=4);
}`
close(fp);
fp=open(raster,0);
for (i=0;i<SIZE;i++) {
    "read(fp, temp, SIZE=4);
    "read(fp, buf2[i], SIZE=4);
    
for (i=0;i<SIZE;i++)
    "for (j=0;j<SIZE;j++) {
    "dt=buf1[i][j];
    "buf1[i][j]=buf2[i][j][1];
    "buf2[i][j][1]=dt;
}

close(fp);
fp=open(raster,0);
fp1=create(temp, PHMODE);
for (i=0;i<SIZE;i++) {
    "write(fp1, temp, SIZE=8);
    "write(fp1, buf2[i], SIZE=4);
}

for (i=0;i<SIZE;i++)
    "read(fp, temp, SIZE=8);
    "for (j=0;j<SIZE;j++) temp[j]=buf1[i][j][3];
    "write(fp1, temp, SIZE=8);

close(fp);
close(fp1);

helpmsg(s)
char *s;
{
printf("Usage: %s raster size
", s);
printf("PPS raster rotation routine. Rotation along the 'diagonal' line.
");
printf("raster .... input datafile
");
printf("size ........ has to be <128x128
");
}