Edge Direction Confidence Estimation for Improvement of Hough Acculations

1987

Eric J. Grajales
University of Central Florida

Find similar works at: https://stars.library.ucf.edu/rtd

University of Central Florida Libraries http://library.ucf.edu

Part of the Engineering Commons

STARS Citation

https://stars.library.ucf.edu/rtd/5060

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
EDGE DIRECTION CONFIDENCE ESTIMATION
FOR IMPROVEMENT OF HOUGH ACCUMULATIONS

BY

ERIC JOSE GRAJALES
B. S. E., University of Central Florida, 1984

THESIS
Submitted in partial fulfillment of the requirements
for the Master of Science in Engineering
in the Graduate Studies Program
of the College of Engineering
University of Central Florida
Orlando, Florida

Spring Term
1987
ABSTRACT

The Hough transform silhouette identification method requires a consistency of edge direction in the identification of similar silhouettes. Many gradient operators used in Hough preprocessing require a thresholding and a non-maxima suppression routine to aid the localization process. These routines may delete edges or cause edge fragmentation. These anomalies degrade the Hough performance due to the lack of accurate silhouette extraction, and reduce the correct localization in the Hough accumulator. Noise or sampling errors can be removed by several enhancement routines presented, they are mean, median, symmetric nearest neighbor, hi pass, and low pass filters.

An edge detection process is presented which produces a directional image and the confidence image that allows subsequent image analysis the ability to determine if the detected edge orientation is accurate, and to what degree. The orientation confidence is produced by comparing a 7 by 7 operator and the Compass Gradient operator. This allows the Hough process the ability to modify the position of accumulation, thereby improving the Hough localization process.
ACKNOWLEDGEMENTS

I would like to thank the Martin Marietta Corporation for their encouragement, and for giving me the time and funding to continue my education. I would also like to thank the personnel in the Image Processing Department, for their understanding and aid throughout the past year, especially to Mr. Albert Sanders and Mr. John Selvege. I would also like to thank Dr. Harley Myler for his helpful insights to some of the problems which I encountered, and for always having the time to help.

To the person who did a wonderful job reproducing my graphs, and printed this report, my brother Nestor, thank you for all your help.

I especially wish to thank my wife Debbie for her encouragement and love all the times I was working late at night, and for whom without I would have never gotten this far. Thank you for all of your help and understanding.
# TABLE OF CONTENTS

ABSTRACT ........................................................................................................................................... ii
ACKNOWLEDGEMENTS ....................................................................................................................... iii
LIST OF FIGURES .................................................................................................................................. v

CHAPTER 1
   INTRODUCTION ................................................................................................................................. 1

CHAPTER 2
   FUNDAMENTALS OF IMAGE PROCESSING ......................................................................................... 5
      Image Detection and Representation ................................................................................................. 5
      Image Processing Terms .................................................................................................................... 7

CHAPTER 3
   IMAGE ENHANCEMENT ...................................................................................................................... 9
      Contrast Enhancement Techniques ................................................................................................... 9
      Image Smoothing ................................................................................................................................ 14
         Spatial Domain Techniques ........................................................................................................ 15
         Frequency Domain Techniques ................................................................................................... 17
      Image Sharpening ............................................................................................................................ 17
         Spatial Domain Techniques ........................................................................................................ 18
         Frequency Domain Techniques ................................................................................................... 18

CHAPTER 4
   EDGE DETECTION ............................................................................................................................... 19
      Gradient Operators ........................................................................................................................... 21
      Second Derivative Operators .......................................................................................................... 26

CHAPTER 5
   PROGRAMMING ENVIRONMENT ...................................................................................................... 29

CHAPTER 6
   DIRECTIONAL CONFIDENCE ............................................................................................................ 32

CHAPTER 7
   CONCLUSION ....................................................................................................................................... 39

APPENDIX A ........................................................................................................................................ 41

BIBLIOGRAPHY ..................................................................................................................................... 66
LIST OF FIGURES

1. Hough R-Table Entries Calculation .................................................. 3
2. Cartesian Coordinates Representation of a Digital Image ...................... 6
3. Real Photograph and Digitized Image .................................................... 6
4. Spatial Coordinates of a 3 by 3 Neighborhood about (x,y) .................... 7
5. Discrete Image Contrast Enhancement Technique .................................. 10
6. Linear Scale Enhancement Technique .................................................. 11
7. Gray Scale Transfer Function .......................................................... 12
8. Histogram Equalization Technique ..................................................... 13
9. Image Averaging Technique .................................................................. 15
10. Symmetric Nearest Neighbor Filtering Technique ............................... 16
11. Two-Dimensional Edge Representation ............................................. 20
12. Roberts Edge Detection Technique .................................................... 22
  (Magnitude Result)
13. Sobel Edge Operator ........................................................................... 23
14. Compass Gradient Edge Detection Technique .................................... 24
15. Canny Edge Detection Technique ....................................................... 27
16. Relation of Amiga Operating System Components ............................. 30
17. Edge Confidence 7 by 7 Operator Results .......................................... 36
  (Magnitude and Directional Images)
18. Edge Confidence Image ....................................................................... 37
CHAPTER 1

INTRODUCTION

The most significant aspect of image analysis is the detection and extraction of regions which describe an image. Images may be analyzed by detecting the intensity changes which separate regions or by dividing the image into regions of similar texture. Once the regions have been detected, they can be analyzed by classical image processing techniques using either structural or statistical methods.

There are several methods for understanding what detected regions represent. Most common statistical methods obtain features about the region, such as area, perimeter, centroids, etc. These features are compared to known region features and the input region is classified with the best match obtained from this process. Structural methods compare a known region to an unidentified region. These techniques include template correlation, Hough transform and others.

In template correlation, a known region is described by a window template. This template is convolved with an input image yielding the position(s) of image regions that may be identical to the known region.
Gonzalez and Wintz (1977) describe this process with the following equation:

\[ R(m,n) = \frac{\sum_{x} \sum_{y} f(x,y) w(x-m,y-n)}{\sum_{x} \sum_{y} f^2(x,y)} \]

where \( f(x,y) \) is the original image value at \( (x,y) \),
\( w(x,y) \) is the known template value at \( (x,y) \),
and \( R(m,n) \) is the correlation image output value at \( (x,y) \)

The Hough transform is a method of representing a desired region silhouette and detecting this representation in an image. The desired silhouette is segmented to obtain an edge direction image. The silhouette is stored by determining its centroid \( (X_c, Y_c) \) and drawing a line to an edge pixel \( (X_i, Y_i) \). A distance from the centroid to the edge pixel and the angle from horizontal to the line connecting the centroid to the pixel position are obtained (Figure 1). A table of all edge pixels, their distance to the centroid, and the angle measured are stored, indexed relative to the edge direction. This table is referred to as an R-Table. This R-Table is a representation of the desired silhouette. To detect this silhouette in an image, the image must be segmented. At each pixel position \( (X_i, Y_i) \) in the segmented image, the possible centers of the silhouette are accumulated in an array \( (A) \) as follows:
The edge direction is determined, and for each R-Table entry with the same edge direction we compute

a. \( X_c = X_i + R_j \cos(\phi_j) \)
\( Y_c = Y_i + R_j \sin(\phi_j) \)

where \( R \) is the distance to the centroid,
\( \phi \) is the angle described earlier,
and \( j \) is the table entry element number.

b. Increment the array \( A \) at position \( (X_c, Y_c) \)

\[ A(X_c,Y_c) = A(X_c,Y_c) + 1 \]

Possible locations of the silhouette are given by the maxima of the array \( A \) (Ballard 1982).

Figure 1. Hough R-Table Entries Calculations.

The Hough transform technique requires that the edge detection be accurate and that the direction of the detected edge be insensitive to light variation. Accuracy in edge detection can be obtained by using a small window operator to detect edges, but these operators are often sensitive to noise and may extract too much information. The Sobel
edge operator is most often used to extract edge information, this edge detector is sensitive to noise, due to its use of a 3 by 3 window operator, and generates a limited number of edge orientations.

A rule based segmentation technique is proposed, which uses a 7 by 7 operator, and a 3 by 3 Compass Gradient operator. A set of criteria will combine the operator outputs, yielding a confidence measure of the accuracy of edge direction. The output directional image can be used to find a desired silhouette within images using the Hough transform.
CHAPTER 2

FUNDAMENTALS OF IMAGE PROCESSING

Image processing often is described as the mathematical analysis and manipulation of digital images by computers. Image processing is used for two major purposes: improvement of images for the human perception (or improving the visual quality of an image) and for "autonomous machine perception" which aids a machine in interpreting, analyzing, and extracting data from images (Gonzalez and Wintz 1977).

Image Detection and Representation

A two-dimensional image is the representation of light that is received from a scene by an optical system. This representation is converted to electrical signals and stored on magnetic media, or it can be recorded by a chemical process as a photograph. For computer processing the image must be converted to digital form.

The picture is converted to digital form by a sampling process. A sampling process will discretize the image in both spatial coordinates and brightness levels. Generally, the spatial distribution of image data is referred to as sampling and the brightness discretization is termed quantization. This digital representation can be described mathematically as a two dimensional function \( f(x,y) \), where \( x \) and \( y \) are the spatial coordinates of the image plane. Figure 2 depicts the cartesian coordinates which are commonly used for spatial
representation. The function values are real, non-negative, and bounded representations of the brightness levels. These brightness levels are usually displayed in monochromatic form (black and white), and are called gray level values. The samples are generally equally spaced in a rectangular array and are quantized to discrete brightness levels (Rosenfeld and Kak 1982). Each element within the sampled array is termed pixel, short for picture element, another term often used is a pel. The result of the sampled and quantized photograph is a digital representation of the scene. Figure 3 shows an original photograph and its corresponding digital representation.

\[(0,0) \rightarrow x \]

\[y \rightarrow (x,y)\]

Figure 2. Cartesian Coordinate Representation of a Digital Image.

Figure 3. Real Photograph and Digitized Image.
Image Processing Terms

Many image processing techniques often refer to a neighborhood of pixels. This is simply a predefined set of pixels about a known pixel location. For example, the spatial coordinates of a pixel and its eight adjacent pixels is shown in Figure 4. This set of pixels is termed a 3 by 3 neighborhood.

\[
\begin{array}{ccc}
(x-1, y-1) & (x,y-1) & (x+1,y+1) \\
(x-1,y) & (x,y) & (x+1,y) \\
(x-1,y+1) & (x, y+1) & (x+1, y+1)
\end{array}
\]

Figure 4. Spatial Coordinates of 3 by 3 Neighborhood about (x,y).

Rosenfeld and Kak (1982) state that operations on digital images fall into three categories: point, local, and geometric operations. For point operations the value of a pixel is dependent on the pixels original value and the operation performed, for example adding a constant to every pixel. In local operations, the pixel value is dependent on the specified neighborhood values about that point, for example, adding the values within a 3 by 3 neighborhood. In geometric operations, the pixel output level depends only on the input level of some other point defined by a geometric transformation. In this case, rotating the image about the origin would be a geometric operation. These operations may be combined to improve the computational analysis of the image.
An alternate method of image analysis is within the Fourier spectrum of an image.

The Fourier transform represents the frequency content of an image, and $F(u,v)$ is defined as the frequency spectrum of $f(x,y)$. The two dimensional discrete Fourier transform of an image is defined by

$$F(f(x,y)) = F(u,v) = \frac{1}{MN} \sum_{y=0}^{N} \sum_{x=0}^{M} f(x,y) \exp[-j2\pi(ux/M+vy/N)]$$

$x = 0,1,2,...,M$
$y = 0,1,2,...,N$

where $f(x,y)$ is the pixel value at $(x,y)$
$F(u,v)$ is the complex Fourier image value at $u,v$

$j = \sqrt{-1}$
$M$ and $N$ are the original image dimensions

and its inverse by

$$f(x,y) = F^{-1}(F(u,v)) = \frac{1}{MN} \sum_{v=0}^{N} \sum_{u=0}^{M} F(u,v) \exp[j2\pi(ux/M + vy/N)]$$

A discussion of the two dimensional Fourier transform, and its properties can be found in (Rosenfeld and Kak 1982), (Gonzalez and Wintz 1977), (Pratt 1978), and other image processing texts.
CHAPTER 3

IMAGE ENHANCEMENT

Digital images may become corrupt or degraded by transmission, sampling, or by conversion from one form to another. Image enhancement techniques improve the visual appearance of an image or "convert the image to a form better suited to human or machine analysis" (Pratt 1978). Image quality is a relative term and is dependent upon the person viewing the image or the final output from an image analysis sequence. Gonzalez and Wintz (1977) divide image enhancement into three categories: contrast enhancement, image smoothing, and image sharpening techniques. There are two processing categories for image enhancement: spatial-domain techniques, which modify the pixels within the image plane and frequency-domain techniques, which modify the frequency spectrum of the image.

Contrast Enhancement Techniques

The contrast manipulation techniques are often used with images whose histogram is skewed or compressed over a small sampling range. Histogram specification techniques modify the histogram of the original image so that it follows some desired form. An image histogram is a function showing, for each gray level, the number of pixels in the image that have that gray level (Pavlidis 1982).
An image may be improved by amplitude rescaling of each pixel. Figure 5 illustrates the discrete image contrast mapping strategy. The output level chosen is that level closest to the exact mapping of an input level. Non-uniform side effects will be caused by this technique. The mathematical transform which can describe the linear scaling process is:

\[
Z = \begin{cases} 
\frac{(Z_k - Z_l) (Z - d) + Z_l}{(e - d)} & \text{for } d \leq Z \leq e \\
Z_l & \text{for } Z < d \\
Z_k & \text{for } Z > e
\end{cases}
\]

This piecewise linear transformation stretches the interval \([d,e]\) between the interval \([Z_l,Z_k]\) (Rosenfeld and Kak 1982). Figure 6 illustrates a low contrast image, and its linearly scaled enhanced image. The scaling mechanism can also be exponential, rayleigh, logarithmic, or gaussian in nature.

Figure 5. Discrete Image Contrast Enhancement.
(Pratt 1978, Redrawn by author)
Figure 6. Linear Scale Enhancement Technique.

(a) Low Contrast Image
(b) Enhanced Image
Assume a variable $r$, which represents the gray level of the pixels within the original image. The pixel values have been normalized, and lie in the region $0 \leq r \leq 1$, where $r = 0$ is dark and $r = 1$ is light. There exists a monotonic point transfer function $s = T(r)$ which produces a level $s$ for every pixel value $r$. This function satisfies two conditions: $T(r)$ is single valued, and $0 \leq T(r) \leq 1$ for $0 \leq r \leq 1$. Figure 7 illustrates the gray level value transformation function. The inverse function, which transforms $s$ back to $r$, is $r = T^{-1}(s)$ for $0 \leq s \leq 1$. This function also satisfies the previous two conditions (Gonzalez and Wintz 1977).

Assume that the original and enhanced images can be characterized by their probability density functions (PDF). The PDF of an image is $p(r_k) = n_k / n$ for $0 \leq r_k \leq 1$, and $k = 0,1,2\ldots$ (the number of gray level values), where $n_k$ is the number of pixels that have the gray level value $r$, and $n$ is the total number of pixels within the image. These
PDFs can be related by the following equation:

$$p_e(s) = \frac{p_o(r)}{dT(r)} \quad r = T^{-1}(s)$$

Equation 1

$P_e(s)$ - pdf of enhanced image
$P_o(s)$ - pdf of original image

(Gonzalez and Wintz 1977).

The most common histogram specification technique is histogram equalization, which assumes the transfer function

$$T(r) = \int p_o(w) \, dw \text{ for } 0 \leq r \leq 1.$$  

The right-hand side of this equation is the cumulative distribution function of the original image. Taking the derivative of $T(r)$ is $d(T(r)) / dr = p_o(r)$. Substituting into equation 1 yields $p_e(s) = p_o(r) / p_o(r) = 1$ for $0 \leq s \leq 1$, a uniformly distributed function (Gonzalez and Wintz 1977). Figure 8 illustrates an original image's PDF, the transfer function, and the resultant enhanced image's PDF.

Figure 8. Histogram Equalization Technique.
(a) Original Image PDF
(b) Transform Function
(c) Enhanced Image PDF

(Gonzalez and Wintz 1977, Redrawn by author)
Other histogram specification transfer functions are described in Pratt, and include exponential, rayleigh, and hyperbolic.

An alternate contrast enhancement scheme was developed by Dorat. His scheme enhances mainly those regions of low contrast, and is insensitive to noise. It uses various histogram parameters, and is applied over a small neighborhood window. The equation which describes this filter is:

\[ c'(x,y) = \begin{cases} 
  Gc(x,y) - \frac{(G-1)W l(x,y)}{W-w(x,y)} & \text{if } w(x,y) \leq G \\
  \frac{(c(x,y) - l(x,y)) W}{w(x,y)} & \text{if } (w(x,y) > G)
\end{cases} \]

where \( c(x,y) \) is the original image value at \((x,y)\)
\( c'(x,y) \) enhanced image value at \((x,y)\)
\( G \) is the gain factor
\( W \) is the maximum value that any pixel may have
\( w(x,y) \) is the width of the local histogram about \( c(x,y) \)
and \( l(x,y) \) is the percentage of the total number of local pixels less than or equal to 1 (Dorat 1982).

**Image Smoothing**

Image smoothing operators primarily eliminate noise or channel transmission errors, while blurring edges and correcting sampling errors (Pratt 1978).
Spatial Domain Techniques

The most often used technique is neighborhood averaging. The pixels within a predefined neighborhood window about \((x,y)\) are averaged, and the pixel value at \((x,y)\) is replaced with this average value. Figure 9 illustrates this technique for a 3 by 3 window.

\[
\begin{array}{ccc}
O1 & O2 & O3 \\
O4 & X & O5 \\
O6 & O7 & O8 \\
\end{array}
\]

\[
\text{if } (X - \frac{1}{8} \sum (O_i) ) > \text{eps then } x = \frac{1}{8} \sum O_i \\
\text{for } i = 1,2,\ldots,8
\]

Figure 9. Image Averaging Technique.

Image averaging is a form of low pass filter. Some other low pass filters' masks are:

\[
H = \begin{pmatrix}
1 & 1 & 1 \\
1 & 2 & 1 \\
1 & 1 & 1 \\
\end{pmatrix}, \quad H = \begin{pmatrix}
1/10 & 1 & 1 \\
1 & 2 & 1 \\
1 & 1 & 1 \\
\end{pmatrix}, \quad H = \begin{pmatrix}
1/16 & 2 & 4 & 2 \\
1 & 2 & 1 \\
\end{pmatrix}
\]

This operation is done for all pixels within the image, except for the boundary pixels. These templates tend to blur the image, and can corrupt edge localization processes.

Another technique is to replace the pixel value at \((x,y)\) with the median value of a set created from the pixel values about a predefined window. This technique is useful in suppressing noise in images, but is better suited to reduce effects of salt and pepper noise than constant additive noise.
Davis et al. describe an alternate smoothing operator, the symmetric nearest neighbor (SNN), that preserves edges and eliminates random noise. It is formally described as follows:

"For a 2n+1 X 2n+1 window centered at the pixel(x,y) in the image, for each pair of pixels

\{(x+I, y+J), (x-I,y-J)\} where -n \leq I, J \leq +n,

select (x+I,y+J) if |G(x,y)-G(x+I,y+J)| < |G(x,y)-G(x-I,y-J)|;
select (x-I,y-J) if |G(x,y)-G(x+I,y+J)| > |G(x,y)-G(x-I,y-J)|;
otherwise select (x,y).

Here, G(P,Q) is the gray value of the pixel (P,Q)" (Davis 1984). From the selected pixel set either the mean or median value of the set replaces the pixel at (x,y). Figure 10 shows a graphical depiction of this process. Davis also describes other smoothing techniques, similar to the SNN, which are the sigma filter, and the K- nearest neighborhood filter.

\[
\begin{align*}
A1 & \quad A2 & \quad B4 \\
A3 & \quad C & \quad B3 \\
A4 & \quad B2 & \quad B1 \\
\end{align*}
\]

\[C = \text{median (minimum } |C- A1, C-B1|)\text{ for } i = 1,2,3,4\]

Figure 10. Symmetric Nearest Neighbor Filter Technique.
Frequency Domain Techniques

Sharp transitions in gray level values in an image cause the high-frequency content in the Fourier transform. An image may be smoothed by attenuating the high-frequency content components of the Fourier spectrum (Gonzalez and Wintz 1977).

The relation $F^{-1}(G(u,v)) = H(u,v)F(u,v)$, where

$G(u,v)$ is the Fourier transform of the resultant image

$H(u,v)$ is the low pass transfer function

and $F(u,v)$ is the Fourier transform of the original image

will generate a smoothed image. Several transfer functions are discussed in Gonzalez and Wintz; they are the ideal low pass filter, the Butterworth low pass filter, and the exponential low pass filter. It should be noted these filters are "zero-phase shift filters which are radially symmetric and can be completely specified by a cross section extending as a function of distance from the origin" of the frequency spectrum (Gonzalez and Wintz 1977).

Image Sharpening

Image sharpening techniques are mainly used for enhancing edges or for increasing the resolution of images. Image sharpening techniques are often referred to as high pass image filters. These techniques tend to attenuate the low-frequency components of an image within the Fourier spectrum.
Spatial Domain Techniques

Edge sharpening can be accomplished by convolving the image with a high-pass mask. Pratt (1978) describes three filter masks:

\[
\begin{align*}
H &= \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad H &= \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \\
H &= \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}
\end{align*}
\]

Gonzales and Wintz (1977) describe an alternate image sharpening technique which is accomplished by differentiation. A gradient operator is applied to an image, which yields the magnitude rate of change within the image, and those values which exceed a given threshold replace the original pixel value with the output magnitude. This will emphasize edges without distorting regions of similar intensity. Another derivative operator, the Laplacian, is described by Rosenfeld and Kak (1982). The Laplacian image is obtained as follows:

\[
\nabla f = \frac{df}{dx} + \frac{df}{dy}
\]

Frequency Domain Techniques

Changes in intensity from one region to another yield the high-frequency information found in the Fourier spectrum. High pass filtering techniques emphasize this high frequency content. The equation used to describe high pass filter techniques is similar to that of low pass filters except that \( H(u,v) \) is a high pass filter transfer function. The counterparts to the low pass filters described earlier are the ideal high pass filter, the Butterworth high pass filter, and the exponential high pass filter.
CHAPTER 4
EDGE DETECTION

Edge detection is probably the most fundamental operation in an image analysis sequence. Pratt defines an edge "as a ramp increase in image amplitude level from a low to a high level" (1978). Figure 11 illustrates an edge in a two-dimensional image. Ballard states that "an edge operator is a mathematical operator (or its computational equivalent) with a small spatial extent designed to detect the presence of a local edge in the image function" (1982). Various edge detection techniques have been described in the computer vision literature, these techniques are often only useful for constrained image characteristics, or for a particular image domain. It should be noted that most edge detection techniques require some noise removal process, to help the detection process.

Joseph Canny describes three attributes edge an detection technique should have, they are:

i. "Good detection. There should be a low probability of failing to mark real edge points, and low probability of falsely marking non-edge points. Since both these probabilities are monotonically decreasing functions of the output signal-to-noise ratio, this criterion corresponds to maximizing signal-to-noise ratio.

ii. Good localization. The points marked by the operator should be as close as possible to the center of the true edge.

iii. Only one response to a single edge. This is implicitly captured in (i) since when two nearby operators respond to the same edge, one of them must be considered a false edge. However, the mathematical form of the first criterion did not capture the multiple response requirement and it had to be made explicit"(Canny 1983).

Canny concludes that optimum edge detection and localization are opposite to one another.
When an intensity change occurs, "there will be a corresponding peak in the first directional derivative, or equivalently, a zero crossing in the second directional derivative of intensity" (Marr and Hildreth 1980). Due to this, Huertas and Medioni (1986) divide edge detectors into two categories: gradient operators and second derivative operators. Gradient operators locate the positions of maximum gradient within the image. Their output may then go thru a non-maximal suppression routine. These thinning routines usually degrade performance, but aid the localization process. Second derivative operators locate the zero-crossings at an edge location. These techniques can have good localization, but their precision depends on the signal-to-noise ratio within the image. Most edge operators compute a magnitude and directional image. The magnitude image describes the maximum slope about each pixel, and the directional image describes the direction perpendicular to the slope.
Gradient Operators

The simplest of the gradient or difference operators is the horizontal and vertical first difference operator. This operator is described as follows:

\[ F_x(i, j) = f(i, j) - f(i-1, j) \]
\[ F_y(i, j) = f(i, j) - f(i, j-1) \]

The output magnitude is calculated by

\[ \text{Mag}(i,j) = \sqrt{(F_x^2 + F_y^2)}. \]

Roberts improved this scheme by making the operator symmetric in the x and y directions, producing the following operator:

\[ F_x(i, j) = f(i, j) - f(i+1, j+1) \]
\[ F_y(i, j) = f(i+1, j) - f(i, j+1) \]

Again the output magnitude is calculated by

\[ \text{Mag}(i,j) = \sqrt{(F_x^2 + F_y^2)} \]

(Hildreth 1980; and Pratt 1978). Figure 12 illustrates the magnitude output from this technique.

The most commonly used and most widely studied edge operator is the Sobel. In it two template mask are convolved over the image. Figure 13 illustrates this operator.
Figure 12. Robert's Edge Detection Technique Magnitude Result.
A0 A1 A2 \quad X = (A2 + 2A3 + A4) - (A0 + 2A7 + A6)  
A7 XX A3  
A6 A5 A4 \quad Y = (A0 + 2A1 + A2) - (A6 + 2A5 + A4)  

The output magnitude result \( G(x,y) = \sqrt{(X^2 + Y^2)} \)  
and the directional orientation \( O(x,y) = \arctan(Y/X) + 90 \)  

Figure 13. Sobel Edge Operator (Pratt 1978).

The directional image is usually quantized to 45 degree increments. Kirsh and Person extended this scheme by convolving the image with four directional template masks, choosing the direction with absolute maximum convolution value (Hildreth 1980). The template masks used are:

<table>
<thead>
<tr>
<th>North-South</th>
<th>East-West</th>
<th>South-East</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 1</td>
<td>-1 0 1</td>
<td>0 -1 -2</td>
</tr>
<tr>
<td>0 0 0</td>
<td>-2 0 2</td>
<td>1 0 -1</td>
</tr>
<tr>
<td>-1 -2 -1</td>
<td>-1 0 1</td>
<td>2 1 0</td>
</tr>
</tbody>
</table>

In both the Sobel and the Kirsh and Persons operators the eight output directions are:

Figure 14 illustrates the output from this technique, the directional image has been quantized to 45 degrees.
Figure 14. Compass Gradient Edge Detection Technique.

(a) Output Magnitude
(b) Output Direction
Many gradient operators do not entirely separate distinct regions, or do not close the gaps between regions. Perkins describes a method which "uses an expansion-contraction technique in which the edge regions are first expanded to close gaps and then contracted after the separate uniform regions have been identified" (Perkins 1980).

Several other gradient operators have been described in the computer vision literature. Of recent development, the entropy operator described in Pratt, and modified by Shiozaki has shown an alternate method of image segmentation. The entropy operator measures the degree of busyness about each pixel. The operator is described as follows:

\[ E(i, j) = \sum_{m=0}^{n} \frac{P(m) \log(P(m))}{\log(n+1)} \]

where

\[ P(m) = \frac{A_m}{\sum_{i=0}^{n} A_i} \]

\( m - 0, 1, 2, ..., 7, 8 \)
\( n - \) number of elements in neighborhood about \((i, j)\)
\( E(i, j) - \) entropy image results
(Shiozaki 1986).
An edge detection technique that has had considerable success is the Canny operator. In his masters thesis, Canny described three basic edge criteria (presented earlier) and define a function which maximizes these criteria. This optimal edge operator is implemented by a simple approximation "in which edges are marked at maxima in gradient magnitude of a gaussian-smoothed image" (Canny 1986). Figure 15 illustrates the Canny edge detection technique, the directions are quantized to 30 degrees.

Second Derivative Operators

Locating the zero-crossings in a Laplacian gaussed image was first suggested by Marr and Hildreth (1980). They describe how the zero-crossings are found by convolving the image with the Laplacian of Gaussian (LoG) operator:

\[
\nabla^2 G = \left( \frac{1}{2\pi(\bar{\sigma})^4} \right) \left( \frac{2 - r^2}{\bar{\sigma}^2} \right) \left( \frac{\exp(-r^2)}{2\bar{\sigma}^2} \right)
\]

where \(\bar{\sigma}\) is the space constant for the gaussian,

and \(r\) is the distance from the center of the operator (Marr and Hildreth 1980).

This operator is circularly symmetric and looks like a mexican-hat. The detection of edges is dependent upon the value of \(\bar{\sigma}\). For high \(\bar{\sigma}\) values only those regions with high contrast edges will be located. This edge operator has been widely studied, and many variations have been developed.
Figure 15. Canny Edge Detection Technique.

a) Magnitude Result
b) Directional Image
An approximation of this technique is the Difference of Gaussians (DOG) function:

$$\nabla^2 (x,y) = G\dot{\Omega}_2(x,y) - G\dot{\Omega}_1(x,y)$$

where $$G\dot{\Omega}(x,y) = \frac{\exp(-(x^2 + y^2))}{2\dot{\Omega}^2}$$

and $$\dot{\Omega}_1$$ and $$\dot{\Omega}_2$$ are again the space constant for the gaussian related by $$\dot{\Omega}_2 / \dot{\Omega}_1 = 1.6$$.

This approximation can be used since the LoG becomes the limit of the DOG as the two sigma values approximate one another (Marr and Hildreth 1980).

These operators have been used successfully in detecting edges to subpixel accuracy (Huertas and Medioni 1986) and edges with high angular localization (Nalwa and Binford 1986).
CHAPTER 5

PROGRAMMING ENVIRONMENT

The Commodore Amiga system provides very powerful capabilities in both hardware and software, including an 880K-byte disk storage, a multitasking operating system, multiple display screens, complex animation capabilities, and the ability to display up to 4,096 colors. The hardware features can be accessed by software through the use of system libraries. These libraries are described in detail in the Amiga ROM Kernel Manual.

In addition to the Motorola 68000 CPU, running at 7.14 MHz, the Amiga has three specialized coprocessors:

1. Copper IC, which controls the graphics system.

2. Sprite Processor IC, which controls the movement of sprites on the graphics display screen (sprites are predefined graphical images easily manipulated about the graphical display).

3. Blitter IC, which moves and combines data from one part of memory to another.

In addition the Amiga has a high quality stereo audio circuit and a built in speech synthesizer. The programmer has direct control over the Amiga special purpose chips, which control the screen resolution, data manipulation, and color selection for each screen. These various hardware features can be readily accessed using various high-level languages. The programming language "C" was chosen for its ability to access complex data structures and its ease of system portability.
The Amiga operating system can be divided into three parts:

1. The executive kernel is the most basic and the most minimal operating system supporting memory allocation, communication between processes and devices, low-level multitasking, and the primary input/output functions.
2. AmigaDOS is a traditional operating system, responsible for the file management system and high-level multiprocessing support.
3. Intuition is the icon-oriented, window-based system that is primary interface for users" (Berry 1986).

The three elements of the Amiga operating system are related as shown in Figure 16.

![Figure 16. Relation of Amiga Operating System Components. (Berry 1986, Redrawn by author)](image)

The Intuition user interface provides the access to system libraries, the selection of screen resolution and characteristics, and the menu structure that was required to access the various software modules. This user interface provided a concise and efficient manner of relating to the user, and aided in the data manipulation of images.

The source for the enhancement and segmentation routine is included in Appendix A. The majority of the program used to evaluate the various enhancement and
segmentation routines consist of Intuition related system calls. These sections are not included in the Appendix.

Due to the complexity of the calculations of the Fourier transform routines and the Marr-Hildreth segmentation routine, described earlier, these routines were not implemented. The Fourier transform of an image size 64 by 64 took approximately 32 minutes to compute, not including the filtering operation and the inverse Fourier. The Marr-Hildreth was approximated by the Difference of Gaussians (DOG). Each gaussian blur requires approximately 12 minutes, for an entire process time of about 28 minutes for an image size 320 by 200. The other processes require approximately 1 to 4 minutes to compute.

The Amiga system can provide an efficient system for image processing. Its ease of programability and the high resolution display graphics are some of the attributes necessary for an image processing workstation. It should be noted that the programs in Appendix A were run on an Amiga with a memory expansion module. Without this expansion the various images manipulated in the program could not have been done.
CHAPTER 6

DIRECTIONAL CONFIDENCE

The Hough transform silhouette identification method requires a consistency of edge direction in the identification of similar silhouettes. Many gradient operators require a thresholding and a non-maxima suppression routine to aid the localization process; however these routines may delete edges or cause edge fragmentation. These anomalies degrade the Hough performance.

The edge detection process presented in this chapter is intended for use on high quality visual imagery. It is assumed that any noise or sampling errors have been removed by previous enhancement routines. The output from the edge detection routines, the directional and confidence image can be used in subsequent image analysis routines to aid the Hough silhouette localization process. Some authors (Sleigh 1986; Canny 1986) have noted that the more information extracted about each position, the more inferences can be determined in subsequent processing. No magnitude image is required for the Hough process. Even though it is not used it should be noted that this is also information about the edge and could be used in future analysis.
Many edge detection techniques are concerned with edge localization and have found many methods of improving this localization. Some techniques include the combination of different template operators in some predefined fashion, while others have devised a rule base systems which infer from various edge information the edge position. The edge and confidence image obtained from this process will allow the subsequent image analysis the ability to determine if the detected edge orientation is accurate and to what degree. This allows the Hough process the ability to modify the position of accumulation described in Chapter 1.

The 7 by 7 operator used to determine the edge orientation was selected because of the high-resolution imagery obtained. A small operator size was affected by small regions and had a limited number of edge directions. The 7 by 7 operator size would not detect small, insignificant regions (2 by 2 and possible 3 by 3 regions), but those edges which separate large distinguishable regions. The size also allowed the determination of up to 8 distinct directions. A direction, in this context, is both the positive and negative orientation of an operator, for example 0 and 180 degrees are both on the same direction. This solved problems which arose with silhouettes that appeared as light regions on dark background and vice-versa. Many edge detection techniques are sensitive to these lighting variations, which cause edge orientations to change 180 degrees. Due to these lighting variations the
Hough accumulator can not locate the silhouette within an image. The increase in edge directions also decreased the computational expense of the Hough process due to the decrease in entries in each of the edge directions indices of the R-Table.

The operator weights were selected so as to detect small amplitude variations within the image. The magnitude and direction from this operator is illustrated in Figure 17.

The eight distinct edge masks used are:

```
0 0 4 6 4 0 0
0 1 1 2 1 1 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 1 0 0 0 -1 0
0 -1 0 0 0 -1 0
0 -1 -1 -2 -1 -1 0
0 0 -4 -6 -4 0 0
0 degree
```

```
0 0 4 1 0 0 0
0 6 1 1 0 0 0
4 1 1 0 0 0 0
1 1 0 0 0 -1 -1
0 0 0 0 -1 -1 -4
0 0 0 -1 -1 -6 0
0 0 0 -1 -4 0 0
45 degrees
```

```
0 0 0 0 0 0 0
0 1 1 0 -1 -1 0
4 1 0 0 0 -1 -4
6 2 0 0 0 -2 -6
4 1 0 0 0 -1 -4
0 1 1 0 -1 -1 0
0 0 0 0 0 0 0
90 degrees
```

```
0 0 6 4 1 0 0
0 4 2 1 1 0 0
1 1 0 0 0 0 0
0 1 0 0 0 -1 0
0 0 0 0 0 -1 -1
0 0 -1 -1 -2 -4 0
0 0 -1 -4 -6 0 0
22.5 degrees
```

```
0 0 1 0 0 0 0
0 4 1 1 0 0 0
6 2 0 0 0 -1 -1
4 1 0 0 0 -1 -4
1 1 0 0 0 -2 -6
0 0 0 -1 -1 -4 0
0 0 0 0 -1 0 0
67.5 degrees
```

```
0 0 0 0 -1 0 0
0 0 0 -1 -1 -4 0
1 1 0 0 0 -2 -6
4 1 0 0 0 -1 -4
6 2 0 0 0 -1 -1
0 4 1 1 0 0 0
0 0 1 0 0 0 0
112.5 degrees
```

```
Additional edge direction information is obtained by determining the compass gradient direction at each pixel position. The two directions, compass gradient and 7 by 7 operator, are compared, and the following criteria determine the confidence image output:

1. If both edge directions are within ±22.5 degrees, the output confidence is a 1 for +22.5 and 2 for -22.5.

2. If both edge directions are within ±45 degrees, the output confidence is a 3 for +45 and 4 for -45.

3. If both edge directions are within ±67.5 degrees, the output confidence is a 5 for +67.5 and 6 for -67.5.

4. If both edge directions are within ±90 degrees, the output confidence is a 7.

Using these criteria the Hough accumulation process has the ability to modify the localization position dependent on the similarity between the two operators. The corresponding confidence image of figure 17 is illustrated in figure 18.
Figure 17. Edge Confidence 7 by 7 Operator Results,

a) Inverse Magnitude Image

b) Directional Image
Figure 18. Edge Confidence Image.
The most accurate confidence would be obtained by comparing many distinct edge
directional operators. This would be computationally expensive and the set of rules could
become unmanageable. Having only two operators reduces the computational expense and
the rule set is simple, yet gives the user enough information for future processing
inferences.
CHAPTER 7

CONCLUSIONS

In conclusion, several distinct silhouette identification methods were illustrated. One of these techniques, the Hough transform, will identify silhouettes within an image by storing a representation of the silhouette in an R-table. With the various components of an image identified, the image may be interpreted by further image analysis.

The processing of images by computers requires images to be converted to digital form. This process has been described, and several image processing functions are illustrated, including neighborhood of pixels, image processing categories, and the Fourier transform.

The most significant areas of image analysis as preprocessing to image understanding are: image enhancement and edge detection. The improvement of images, either for computational analysis or for the human visual system is termed image enhancement. There exist various image enhancement algorithms in the computer vision literature. A brief description of some current algorithms were presented in Chapter 3 for both for spatial and frequency domain techniques. Edge detection operators are a subject of much discussion in the computer vision literature. Edge detection operators should have
the three attributes Canny (1983) described. Several edge detectors are described and their outputs illustrated, among them the Roberts operator, the Sobel edge detection technique, and the Canny operator.

It was shown that the Hough transform method can be improved by the addition of an edge direction confidence image. A 7 by 7 operator was described which quantized the directional image into 8 distinct directions and aided in the localization of small regional intensity differences. The confidence image is obtained by comparing two distinct edge directional operators. With this image the Hough localization process may be modified based on the accuracy of edge orientation.
#include <exec/types.h>
#include <intuition/intuition.h>

extern int SepAPen(), WritePixel(), ReadPixel();
extern int SetRast(), ClipBlit();
extern int Permit(), Forbid();

void readin(), end_filt(), re_read_img();

static short int im9_in[320][200];
static BOOL read_in = FALSE;

mean()
{
    ***********************************************
    * This subroutine will compute the mean of a *
    * 3 by 3 neighborhood, and replace the mean *
    * value in the center pixel. *
    ***********************************************

    extern struct Window *w, *BackWind;
    unsigned short int x, y, times;
    int temp;
    short int i, j;

    Forbid();
    /* Check to see if image has already been
    read into processing array */
    if (!read_in)
        readin()
        read_in = TRUE;
    }
    for(y = 12; y < 198; y++)
        for(x = 1; x < 318; x++)
            /* for every pixel element*/
            temp = 0;
            times = 0;
        /* Sum all the pixels in the 3 by 3 neighborhood in temp*/
        for(j = -1; j < 2; j++)
            for(i = -1; i < 2; i++)
                if(j != 0 && i != 0)
                    temp += im9_in[x-i][y-j];
                times += 1;
        }
    }
    /* Display output value in enhanced window */
    SetAPen(BackWind->RFort,
        (int) ((float) temp /
        (float) times + 0.5));
WritePixel(BackWind->RPort,x,y);
}
/* Call clean up routine */
end_filt();
return(NULL);
}
median()
{
/**********************************************
* This subroutine will compute the median of a*
* 3 by 3 neighborhood, and replace the mean *
* value in the center pixel. *
***********************************************/
extern struct Window *w, *BackWind;
unsigned short int x, y, times;
int temp, a[9];
short int i, j;
Forbid();
/* Check to see if image has already been
read into processing array */
if (!read_in){
  readin();
  read_in = TRUE;
}
for(y = 12; y < 198; y++) { /* for every line element */
  for(x = 1; x < 318; x++) { /* for every pixel element*/
    temp = 0; times = 0;
    /* Input elements of 3 by 3 neighborhood into array a */
    for(j = -1; j < 2; j++) {
      for(i = -1; i < 2; i++) {
        if(j != 0 && i != 0) {
          a[temp] = img_in[x-i][y-j];
          temp++; times += 1;
        }
      }
    }
    /* Sort array a */
    for(j = 0; j < times; j++) {
      for(i = j+1; i < times; i++) {
        if(a[i] < a[j]) {
          temp = a[i];
          a[i] = a[j];
          a[j] = temp;
        }
      }
    }
  }
}
}
a[j] = temp;
}
}
times /= 2;
/* Display median value of array a in enhanced window */
SetAPen(BackWind->RPort, (int)
  ((float)(a[times-1]+a[times])/2.0-0.5));
WritePixel(BackWind->RPort, x, y);
}
/* Call clean up routine */
end_filt();
return (NULL);
}
snn()
{
/*****************************
* This subroutine will compute the SNN of a *
* 3 by 3 neighborhood, and replace the median*
* value of array d in the center pixel.      *
******************************/
extern struct Window *w, *BackWind;
unsigned short int x, y, times;
int c, a, b, d[9];
int temp, temp1;
short int i, j;
Forbid();
/* Check to see if image has already been
read into processing array */
if (!read_in){
  readin();
  read_in = TRUE;
}

for(y = 12; y < 198; y++){
  /* for every line element */
  for(x = 1; x < 318; x++){
    /* for every pixel element*/
    temp = 0;
    times = 0;
    c = img_in[x][y] * 2;
    /* for 3 by 3 select the opposite pairs about the center
    and check to see which one is closer to center value */
    for(j = 0; j < 2; j++){
      for(i = -1; i < 2; i++){
        if(j != 0 && i != 0){
a = img_in[x + i][y - j];
b = img_in[x - i][y + j];
templ = a + b;
if ( templ > c) {
    if ( a > b)
        d[temp] = b;
    else
        d[temp] = a;
} else if ( templ < c) {
    if ( a > b)
        d[temp] = a;
    else
        d[temp] = b;
} else d[temp] = c / 2;
temp++; times += 1;

/* Sort the selected values of array d */
for (j = 0; j < times; j++) {
    for (i = j + 1; i < times; i++){
        if (d[i] < d[j]){
            templ = d[i];
            d[i] = d[j];
            d[j] = templ;
        }
    }
}
times /= 2;

/* Display output median value of array d in enh. window */
SetAPen(BackWind->RPort, (int)
    ((float)(d[times-1]+d[times])/2.0-0.5));
WritePixel(BackWind->RPort, x, y);
}

} /* Call clean up routine */
end_filt();
return(NULL);

/* Frequency filter in spatial domain */
lopass()
{
/*
 **********************************************
 * This subroutine will compute the lopass    *
 * filter of a 3 by 3 neighborhood, and      *
 * replace the convolution sum value in the   *
 * center pixel.                             *
 ***********************************************/

extern struct Window *w, *BackWind;
unsigned short int x, y;
int temp;
short int i, j;
static short int lomask[3][3] = {
{1, 2, 1},
{2, 4, 2},
{1, 2, 1}
};

Forbid();
/* Check to see if image has already been
read into processing array */
if (!read_in){
    readin();
    read_in = TRUE;
}
for(y = 12; y < 198; y++){    /* for every line element */
    for(x = 1; x < 318; x++){  /* for every pixel element*/
        temp = 0;
        /* Apply convolution mask */
        for(j = -1; j < 2; j++){
            for(i = -1; i < 2; i++){
                temp += lomask[i+1][j+1] * img_in[x+i][y+j];
            }
        }
        /* Display output value in enhanced window */
        SetAPen(BackWind->RPort,
        (int) ((float) temp/16.0));
        WritePixel(BackWind->RPort,x,y);
    }
}
/* Call clean up routine */
end_filt();
return(NULL);
hipass()
{
/*****************************/
*  This subroutine will compute the hipass  *
*  filter of a 3 by 3 neighborhood, and   *
*  replace the convolution sum value in the *
*  center pixel.                          *
/*****************************/

extern struct Window *w, *BackWind;
unsigned short int x, y;
int temp;
short int i, j;
static short int himask[3][3] = {
  { 0, -1,  0},
  {-1,  5, -1},
  { 0, -1,  0}   
};

Forbid();
/* Check to see if image has already been 
  read into processing array */
if (!read_in){
  readin();
  read_in = TRUE;
}
for(y = 12; y < 198; y++) {
  /* for every line element */
  for(x = 1; x < 318; x++) {
    /* for every pixel element*/
    temp = 0;
    /* Apply convolution array */
    for(j = -1; j < 2; j++) {
      for(i = -1; i < 2; i++) {
        temp += himask[i+1][j+1] *
          img_int[x+i][y+j];
      }
    }
    if (temp > 15) temp = 15;
    /* Display output value in enhanced window */
    SetAPen(BackWind->RPort, (int) temp);
    WritePixel(BackWind->RPort, x, y);
  }
}
/* Call clean up routine */
end_filt();
return(NULL);
}
void readin()
{
/*****************************************************************************/
* This subroutine will read input image into *
* the processing array                 *
****************************************************************************/

extern struct Window *w, *BackWind;
unsigned short int x, y;
int a;

for(y = 11; y < 200; y++){
    for(x = 0; x < 320; x++){
        a = (int) ReadPixel(w->RPort, x, y);
        if(a >= 0) img_in[x][y] = (unsigned short int) a;
        else img_in[x][y] = 0;
    }
}

void end_filt()
{
/*****************************************************************************/
* This subroutine will move the enhance image* 
* so the user may view it, and it will clear * 
* the backwindow for future processing.     *
****************************************************************************/

extern struct Window *w, *BackWind;

ClipBlit(BackWind->RPort, 0, 11,
    w->RPort, 320, 11, 320, 189, 0x0C0);
ClipBlit(BackWind->RPort, 0, 11,
    BackWind->RPort, 0, 11, 320, 189, 0x40);
Permit();
}

void re_read_img()
{
/*****************************************************************************/
* This subroutine will allow new images to be* 
* read into the processing array.    *
****************************************************************************/

read_in = FALSE;
##include <exec/types.h>
#include <intuition/intuition.h>
#include <math.h>

extern int SetRast(),ClipBlit(), SetAPen();
extern int WritePixel(), ReadPixel();
extern int Permit(),Forbid();

extern int gauss();

void readseg();

static short int img_seg[320] [200];

Compass_Gradient()
{
    /* This routine will compute the compass Gradient edge
detection technique. */
    extern struct Window *w, *BackWind;
    unsigned short int x, y, k;
    short int i, j;

    /* Define the various convolution masks */
    static short int mask1[3] [3] = {
        {1, 2, 1},
        {0, 0, 0},
        {-1, -2, -1}};
    static short int mask2[3] [3] = {
        {-2, -1, 0},
        {-1, 0, 1},
        {0, 1, 2}};
    static short int mask3[3] [3] = {
        {-1, 0, 1},
        {-2, 0, 2},
        {-1, 0, 1}};
    static short int mask4[3] [3] = {
        {0, 1, 2},
        {-1, 0, 1},
        {-2, -1, 0}};

    int result[4], max1, dir;

    /* Read image to process */
    readseg();
    /* Clear display */
    SetRast(w->RPort,0);

    for(y = 11; y < 199; y++){
        for(x = 1; x < 320; x++){
            ..
for(k = 0; k < 4; k++)
    result[k] = 0;

/* Apply convolution masks */

for(j = -1; j < 2; j++)
    for(i = -1; i < 2; i++)
        result[0] += (mask1[i+1][j+1] *
                       img_seg[x+i][y+j]);
    result[1] += (mask2[i+1][j+1] *
                       img_seg[x+i][y+j]);
    result[2] += (mask3[i+1][j+1] *
                       img_seg[x+i][y+j]);
    result[3] += (mask4[i+1][j+1] *
                       img_seg[x+i][y+j]);

max1 = -1;
dir = 0;

/* Determine which mask contains the largest magnitude */

for(k = 1; k < 5; k++)
    if( abs(result[k-1]) > max1)
        max1 = abs(result[k-1]);
    dir = (int) k;

if(max1 > 15) max1 = 15;
if(result[dir] < 0) dir += 4;

/* Display magnitude and directional images */

SetAPen(w->RPort, max1);
WritePixel(w->RPort, x, y);
if(max1 > 0)
    SetAPen(w->RPort, dir);
    WritePixel(w->RPort, x+320, y);

return(NULL);
roberts()
{
/* Subroutine to compute the Roberts edge detection
   magnitude image */
extern struct Window *w;
unsigned short int x, y;
short int i, j;

int maxl;

readseg();

for(y = 11; y < 199; y++){
    for(x = 0; x < 319; x++){

        /* Compute the components necessary for Roberts Operator */
        i = img_seg[x] [y] - img_seg[x+1] [y+1];
        j = img_seg[x+1] [y] - img_seg[x] [y+1];

        /* Determine Magnitude image value */
        maxl = (int) (sqrt( (double) (i*i + j*j) ) );
        maxl = abs(maxl/2);
        if(maxl > 15) maxl = 15;

        /* Display Magnitude image */
        SetAPen(w->RPort, maxl);
        WritePixel(w->RPort, x+320, y);
    }
}

return(NULL);
}
canny(pm)
int *pm;
{
/* Subroutine to Compute the CANNY edge
detection algorithm */

extern struct Window *w, *BackWind;
unsigned short int x, y, k, m, n;
short int i, j;

/* Define the various convolution masks */

static short int mask1[5][5] = {
  {0, 0, 0, 0, 0},
  {1, 4, 6, 4, 1},
  {0, 0, 0, 0, 0},
  {-1, -4, -6, -4, -1},
  {0, 0, 0, 0, 0}
};

static short int mask2[5][5] = {
  {0, 0, 0, 4, 1},
  {0, 4, 6, 0, 0},
  {1, 0, 0, 0, -1},
  {0, 0, -6, -4, 0},
  {-1, -4, 0, 0, 0}
};

static short int mask3[5][5] = {
  {0, 0, 1, 0, -1},
  {0, 4, 0, 0, -4},
  {0, 6, 0, -6, 0},
  {6, 0, 0, -4, 0},
  {1, 0, -1, 0, 0}
};

static short int mask4[5][5] = {
  {0, 1, 0, -1, 0},
  {0, 4, 0, -4, 0},
  {0, 6, 0, -6, 0},
  {0, 4, 0, -4, 0},
  {0, 1, 0, -1, 0}
};

static short int mask5[5][5] = {
  {1, 0, -1, 0, 0},
  {4, 0, 0, -4, 0},
  {0, 6, 0, -6, 0},
  {0, 4, 0, 0, -4},
  {0, 0, 1, 0, -1}
};

static short int mask6[5][5] = {
  {-1, -4, 0, 0, 0},
  {0, 0, -6, -4, 0},
  {1, 0, 0, 0, -1},
  {0, 4, 6, 0, 0},
  {0, 0, 0, 4, 1}
}
int result[6], max1, dir;

/* Filter the image with a Gaussian operator */
dogauss( (float) *pm / 100.0);
ClipBlit(w->RPort, 320, 11, w->RPort, 0, 11, 320, 189, 0x0C0);
ClipBlit(w->RPort, 320, 11, w->RPort, 320, 11, 320, 189, 0x40);

/* Read image into processing array and clear display */
readseg();
SetRast(w->RPort, 0);

for(y = 12; y < 198; y++){
    for(x = 2; x < 318; x++){
        for(k = 0; k < 6; k++)
            result[k] = 0;
        n = 0;
        for(j = -2; j < 3; j++){
            m = 0;
            for(i = -2; i < 3; i++){
                /* Apply the various convolution mask */
                result[0] += (mask1[m] [n] * img_sea[x+i] [y+j]);
                result[1] += (mask2[m] [n] * img_sea[x+i] [y+j]);
                result[2] += (mask3[m] [n] * img_sea[x+i] [y+j]);
                result[3] += (mask4[m] [n] * img_sea[x+i] [y+j]);
                result[4] += (mask5[m] [n] * img_sea[x+i] [y+j]);
                result[5] += (mask6[m] [n] * img_sea[x+i] [y+j]);
                m++;
            }
            n++;
        }
    }
}

max1 = -1;
dir = 0;
/* Determine which convolution mask had the largest output*/
for(k = 1; k < 7; k++){
    if( abs( result[k-1] ) > max1){
max1 = abs(result[k-1]);
dir = (int) k;
}
max1 /= 16;
if(max1 > 15) max1 = 15;

/* Display magnitude and directional images */

if( max1 > 0 ){
    SetAPen(w->RPort, max1);
    WritePixel(w->RPort, x, y);
    SetAPen(w->RPort, dir);
    WritePixel(w->RPort, x+320, y);
}
return(NULL);

}
diff_of_gauss(pm, pn)
int *pm, *pn; /* gauss values for 1st and 2nd image */
{
    extern struct Window *w, *BackWind;

    /* readseg(); */
    /* dogauss((float) *pm / 100.0); */
    /* move image to backwindow enhanced */
    /* dogauss((float) *pn / 100.0); */
    /* subtract images */

    /* find the zero crossings */
    return(NULL);
}

entropy()
{
    /* Entropy operator edge detection technique */
    extern struct Window *w, *BackWind;
    unsigned short int x, y, k;
    short int i, j;
    short int output[200][320];
    float result[9], max1, min1;

    readseg();
    for(y = 11; y < 199; y++) {
        for(x = 1; x < 320; x++) {
            /* Clear processing array */
for(k = 0; k < 9; k++)
result[k] = 0;
max1 = 0;

/* Calculate mean of 3 by 3 */
for(j = -1; j < 2; j++)
  for(i = -1; i < 2; i++)
    result[0] += (float) img_seg[x+i] [y+j];
result[0] /= 9.0;
result[0] = (float) img_seg[x] [y] / result[0];
result[1] = (float) img_seg[x-1] [y-1] / result[0];
result[2] = (float) img_seg[x] [y-1] / result[0];
result[3] = (float) img_seg[x+1] [y-1] / result[0];
result[4] = (float) img_seg[x-1] [y] / result[0];
result[5] = (float) img_seg[x] [y] / result[0];
result[6] = (float) img_seg[x+1] [y] / result[0];
result[7] = (float) img_seg[x-1] [y+1] / result[0];
result[8] = (float) img_seg[x] [y+1] / result[0];

for(k = 0; k < 9; k++)
  max1 += result[k] * log( (double) result[k] ) /
           log(10.0); 
output [x-100][y-75] = (int) -(max1 + 0.5);

/* Scale resultant image */
max1 = -1000.0;
min1 = 1000.0;
for(y = 0; y < 50; y++)
  for(x = 0; x < 100; x++)
    max1 = max(max1, output[x] [y]);
  min1 = min(min1, output[x] [y]);

max1 = max1 - min1;
for(y = 11; y < 199; y++)
  for(x = 1; x < 320; x++)
    /* Display magnitude image */
    SetAPen(w->RPort, 1+ (int) ( (14.0 *
                                     (float) (output[x-100] [y-75]-min1) )/
                                     (float) max1 + 0.5 ) );
    WritePixel(w->RPort, x+320, y);
}
return(NULL);
seg_fus()
{
/* Edge Detection technique that combines the Compass Gradient and a 7 by 7 Convolution Masks. */
extern struct Window *w, *BackWind;
unsigned short int x, y, k, m, n;
short int i, j;

/* Define the various convolution masks */

static short int cg1[7][7] = {
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 1, 2, 1, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, -1, -2, -1, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 }
};

static short int cg2[7][7] = {
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 2, 1, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, -1, -2, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 }
};

static short int cg3[7][7] = {
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 1, 0, -1, 0, 0 },
    { 0, 0, 2, 0, -2, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 }
};

static short int cg4[7][7] = {
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, -1, -2, 0, 0 },
    { 0, 0, 1, 0, -1, 0, 0 },
    { 0, 0, 2, 1, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0, 0 }
};

static short int mask1[7][7] = {
    { 0, 0, 4, 6, 4, 0, 0 },
    { 0, 1, 1, 2, 1, 1, 0 },
    { 0, 0, 0, 0, 1, 0 },
    { 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0, 0 }
};
static short int mask2[7] [7] =
{ { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 } }

static short int mask3[7] [7] =
{ { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 } }

static short int mask4[7] [7] =
{ { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 } }

static short int mask5[7] [7] =
{ { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 } }

static short int mask6[7] [7] =
{ { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 } }
static short int mask7[7][7] =
{ { 0, 0, 0, -1, -4, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 1, 1, 1, 0, 0, 0, 0 },
  { 4, 1, 1, 0, 0, 0, 0 },
  { 0, 6, 1, 1, 0, 0, 0 },
  { 0, 0, 4, 1, 0, 0, 0 } }; 

static short int mask8[7][7] =
{ { 0, 0, -1, -4, -6, 0, 0 },
  { 0, 0, -1, -1, -2, -4, 0 },
  { 0, 0, 0, 0, 0, 0, 0 },
  { 0, 1, 1, 0, 0, 0, 0 },
  { 1, 1, 0, 0, 0, 0, 0 },
  { 0, 4, 2, 1, 1, 0, 0 },
  { 0, 0, 6, 4, 1, 0, 0 } }; 

int cgresult[4], cgmax1, cgdir;

int result[8], max1, dir;

/* read image into processing array and clear display */
readseg();
SetRast(w->RPort, 0);

for(y = 13; y < 197; y++)
  for(x = 3; x < 317; x++)
    /* Clear convolution result arrays */
    for(k = 0; k < 8; k++)
      result[k] = 0;
    for(k = 0; k < 4; k++)
      cgresult[k] = 0;

    /* Apply Convolution masks */
    n = 0;
    for(j = -3; j < 4; j++)
      m = 0;
      for(i = -3; i < 4; i++)
        cgresult[0] += (cg1[m][n] * 
                       img_seg[x+i][y+j]);
        cgresult[1] += (cg2[m][n] * 

\[ \text{result}[0] \quad \text{result}[1] \quad \text{result}[2] \quad \text{result}[3] \quad \text{result}[4] \quad \text{result}[5] \quad \text{result}[6] \quad \text{result}[7] \]

\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
\[ \text{img}_\text{seg}[x+i][y+j] ; \]
if (cmax1 > 15) cmax1 = 15;

if (max1 > 0) {
    /* Display magnitude and directional images */
    SetAPen(w->RPort, (max1+cmax1)/2);
    WritePixel(w->RPort, x, y);
    SetAPen(w->RPort, dir);
    WritePixel(w->RPort, x+320, y);

    /* Determine confidence and display confidence image */
    switch (cgdir) {
        case 1:
            if ((dir == 1) || (dir == 2) || (dir == 8)) {
                SetAPen(BackWind->RPort, 1);
                WritePixel(BackWind->RPort, x, y);
            }
            else if ((dir == 3) || (dir == 7)) {
                SetAPen(BackWind->RPort, 3);
                WritePixel(BackWind->RPort, x, y);
            }
            else if ((dir == 4) || (dir == 5)) {
                SetAPen(BackWind->RPort, 5);
                WritePixel(BackWind->RPort, x, y);
            }
            else {  
                SetAPen(BackWind->RPort, 7);
                WritePixel(BackWind->RPort, x, y);
            }
            break;
        case 2:
            if ((dir == 2) || (dir == 3) || (dir == 4)) {
                SetAPen(BackWind->RPort, 1);
                WritePixel(BackWind->RPort, x, y);
            }
            else if ((dir == 1) || (dir == 5)) {
                SetAPen(BackWind->RPort, 3);
                WritePixel(BackWind->RPort, x, y);
            }
            else if ((dir == 8) || (dir == 6)) {
                SetAPen(BackWind->RPort, 5);
                WritePixel(BackWind->RPort, x, y);
            }
            else {  
                SetAPen(BackWind->RPort, 7);
                WritePixel(BackWind->RPort, x, y);
            }
            break;
    }
}
break;
case 3:
    if ( (dir == 4) || (dir == 5) || (dir == 6) ) {
        SetAPen(BackWind->RPort, 1);
        WritePixel(BackWind->RPort, x, y);
    }
    else if ( (dir == 3) || (dir == 7) ) {
        SetAPen(BackWind->RPort, 3);
        WritePixel(BackWind->RPort, x, y);
    }
    else if ( (dir == 2) || (dir == 8) ) {
        SetAPen(BackWind->RPort, 5);
        WritePixel(BackWind->RPort, x, y);
    }
    else {
        SetAPen(BackWind->RPort, 7);
        WritePixel(BackWind->RPort, x, y);
    }
    break;
case 4:
    if ( (dir == 6) || (dir == 7) || (dir == 8) ) {
        SetAPen(BackWind->RPort, 1);
        WritePixel(BackWind->RPort, x, y);
    }
    else if ( (dir == 5) || (dir == 1) ) {
        SetAPen(BackWind->RPort, 3);
        WritePixel(BackWind->RPort, x, y);
    }
    else if ( (dir == 4) || (dir == 2) ) {
        SetAPen(BackWind->RPort, 5);
        WritePixel(BackWind->RPort, x, y);
    }
    else {
        SetAPen(BackWind->RPort, 7);
        WritePixel(BackWind->RPort, x, y);
    }
    break;
default:
    SetAPen(BackWind->RPort, 15);
    WritePixel(BackWind->RPort, x, y);
} /* switch cgdir */
} /* max1 > 0 */

return(NULL);
void readseg()
{
    /* Read image into processing array */
    extern struct Window *w;
    unsigned short int x, y;
    int a;

    for(y = 11; y < 200; y++)
        for(x = 0; x < 320; x++)
            a = (int) ReadPixel(w->RPort, x, y);
            if(a >= 0) img_seg[x][y] = (unsigned short int) a;
            else img_seg[x][y] = 0;
}
#include <exec/types.h>
#include <intuition/intuition.h>
#include <math.h>

extern int SepAPen(), WritePixel(), ReadPixel();
extern int SetRast(), ClipBlit();
extern int Permit(), Forbid();

void readgauss(), gausx(), gausy();

static float operator [256];
static float img_gauss[320] [200];
static float img_out[320] [200];

do9auss(sigma)
float sigma;
{
    extern struct Window *w, *BackWind;
    unsigned short int x, y;
    /* Read image into processing array and apply gaussian convolution first in x direction then in y direction */
    readgauss();
    gausx(sigma);
    gausy(sigma);
    /* Display Gaussian image */
    for(y = 11; y < 199; y++){
        for(x = 1; x < 318; x++){
            if(img_gauss[x][y] > 15) img_out[x][y] /= 2.0;
            SetAPen(BackWind->RPort, 
                (int) ( img_out[x][y] + 0.5 ));
            WritePixel(BackWind->RPort,x,y);
            }
        ClipBlit(BackWind->RPort,0,11,
            w->RPort,320,11,320,189,0x0C0);
        ClipBlit(BackWind->RPort,0,11,
            BackWind->RPort,0,11,320,189,0x40);
        return(NULL);
    }
}

float gauss(sigma, dist_sq)
float sigma, dist_sq;
{
    /* Function that return the Gaussian value at specified distance */
    float output;
    float sqrt_2_pi = 2.506628;
return (1.0 / (sigma * sqrt_2_pi) ) *
(float) exp(-0.5 *
(double)dist_sq/(pow( (double)sigma, 2.0) ) );
}
int gengauss(sigma)
float sigma;
{
/* Subroutine that determines operator size and the values
 of the convolution mask */
unsigned short int i;
float sum;
int gauss_size;

operator[0] = gauss(sigma,0.0);
sum = operator[0];
for(i = 1; i < 256; i++){
    operator[i] = gauss(sigma, (float) (pow( (double) i , 2.0) ) );
    sum += operator[i] + operator[i];
    if( (operator[i] / operator[0]) < 0.022){
        gauss_size = i - 1;
        for(i = 0; i < gauss_size; i++)
            operator[i] = operator[i] / sum;
        return(gauss_size);
    }
}

gauss_size = 256;
for(i = 0; i < gauss_size; i++)
    operator[i] = operator[i] / sum;
return(gauss_size);
}

void gaussx(sigma)
float sigma;
{
/* Apply gaussian mask in x direction */
unsigned short int i,j;
short int k;
unsigned short int opsize;
float mini, maxi;

opsize = gengauss(sigma);
for(i = 11; i < 197; i++){
    for(j = 1; j < 319; j++){
        img_out[j] [i] = 0.0;
        mini = max(1, j - opsize) - j;
        maxi = min(320, j + opsize) - j;
        for(k = mini; k < maxi+1; k++){
void gausy(sigma)
float sigma;
{
    /* Apply Gaussian mask in y direction */
    unsigned short int i, j;
    short int k;
    unsigned short int opsize;
    float min1, max1;
    opsize = gengauss(sigma);
    for(j = 1; j < 319; j++){
        for(i = 11; i < 199; i++){
            img_gauss[j][i] = 0.0;
            min1 = max(1, i - opsize) - i;
            max1 = min(200, i + opsize) - i;
            for(k = min1; k < max1+1; k++){
                img_gauss[j][i] = img_gauss[j][i] +
                                 img_out[j][k+i] * operator(abs(k));
            }
        }
    }
}

void readgauss()
{
    /* Read image into processing array */
    extern struct Window *w, *BackWind;
    unsigned short int x, y;
    int a;
    for(y = 11; y < 199; y++){
        for(x = 1; x < 319; x++){
            a = (int) ReadPixel(w->RPort,x,y);
            if(a >= 0) img_gauss[x][y] = (float) a;
            else img_gauss[x][y] = 0.0;
        }
    }
}
BIBLIOGRAPHY


