USING THE HOUGH TRANSFORM FOR ANALYSIS OF IMAGES CONTAINING STRAIGHT LINES

By Andrew Francis Dachs

Submitted to the University of Cape Town in partial fulfillment of the requirements for the degree of Master of Science in Engineering

Cape Town, September 1990

DECLARATION

I declare that this dissertation is my own unaided work. It is being submitted for the degree of Master of Science in Engineering at the University of Cape Town. It has not been submitted before for any degree or examination at this or any other university.

Signed by candidate

A. F. Dachs
(Signature of Candidate)

30th of September, 1990.

ABSTRACT

The Hough transform is a means for finding straight lines in an image. Since it is robust and efficient it is widely used in machine vision systems. The Hough transform has been shown to be a special case of the Radon transform. As a result, the Hough transform can be inverted using the inverse Radon transform. The Radon transform is important in the medical field, where its inverse, reconstruction from projections, is used to view "slices" through a patient in Computer Aided Tomography.

The straight line Hough transform produces a two dimensional parameter space. A straight line in the image produces a peak in this space. Normally, the Hough transform extracts two parameters for each line in the image. Two parameters can describe a line mathematically, but a line segment requires four parameters since the end points must be defined. It is possible to avoid extending the Hough space to four dimensions and still extract line segments. The method presented here achieves this by filtering the two dimensional Hough space before inversion with the inverse Radon transform.

The Hough, Radon and inverse Radon transforms are implemented on general purpose computers and the different algorithms for inverting the Radon transform are discussed.

The "filtering in Hough space" method is applied to the problem of extracting polygons, or polyhedra, from images. The information extracted can be used by a Computer Aided Design (CAD) system to model the scene. Other uses of the forward transform / filter / inverse transform method are discussed. For example, linear features in images can be enhanced in this manner.

This method can be used in a machine vision system in which straight lines must be extracted from an image. However, the computation times are too long for a real time system. In this case dedicated hardware would be required. Such dedicated hardware has been described in the literature.

It is possible to extend the Hough transform to other parametric curves, for example circles and ellipses. However, no inverse transform exists for these extensions. Therefore the filtering technique is limited to linear features at this stage.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Professor Gerhard De Jager, for his guidance and unfailing enthusiasm throughout this project.

I would also like to thank:

Cornelius Van Rènsburg for his noise measurement programs.

Professor Mike Inggs for making the Synthetic Aperture Radar images available.

Gabi Wehnert deserves special thanks for her support and help, particularly with the word processing.

I am also grateful to the CSIR for their financial support over the past two years.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

List of Figures:

List of Tables:

Page Number

CHAPTER 1

1. Introduction

Industrial robots are used on many assembly lines throughout the world. The environment that the robots work in is generally well defined. The robot might reach for a part, which must be in a known position and orientation, and weld it to a car body, which is also in a known position and orientation. This type of robot is extremely useful and economic for large scale production. However, for smaller scale production, they tend to be too inflexible.

Machine vision adds this flexibility, giving robots the capability to "see". These robots with machine vision are making the robots without senses obsolete (Kent, 1986). The largest application of such robots is in automated inspection, for example the inspection of printed circuit boards. These robots can inspect similar objects continuously without becoming bored or distracted. They can be tailored to different tasks by changing the software, while the hardware of the robot remains the same.

Machine, or computer, vision is the process of extracting information about a scene by analysing images of that scene (Rosenfeld, 1988). Such images can be created in a variety of ways, such as radio astronomical maps, infrared images or those formed by visible light. To process an image with a digital computer it must be converted into an array of pixel (picture elements) values. In most systems the intensity of the light is measured at a set of rectangularly spaced sample points. This gives a two dimensional array of values representing the gray level at each point. One or more such images form the input to a machine vision system. The output is information about the scene that gave rise to the images. For the assembly line robot the information could be the position and orientation of the next part it should pick up. In general the aim would be to recognise various objects that may be present in the scene.

The typical hardware for such a system would be: a camera with a frame capture device that stores the image as digital numbers, a computer for processing the information, and a device which acts on the result such as a robot arm (see figure 1.1). The requirements for the vision system vary greatly depending on the application. One of the largest factors is the resolution of the camera. Typically the camera would produce an array of 256 by 256,

512 by 512, or 1024 by 1024 pixels. An array of 512 by 512 pixels will adequately represent the monochrome image from a commercial television or video recorder if each pixel can take on one of 256 gray levels. The amount of information to be processed increases rapidly as the resolution increases. The computing power required, and cost, increases accordingly. Therefore, one of the objectives of a machine vision system must be to reduce the amount of data. A second major consideration is the response time required. Some applications do not require a real time system. In this case a sample can be taken and analysed off-line by a cheap computer while the production continues. On the other hand, if a decision must be made within a fraction of a second, extremely specialised and expensive computers will be required. A third consideration is the number of bits used to represent the luminance of each pixel. In a gray level image each pixel, typically, may take on one of 256 levels. This requires eight bits, or one byte, per pixel. Some systems use only one bit per pixel. In these systems a pixel may be only light or dark. Although these systems are generally faster (since there is less information) they are extremely dependant on lighting conditions. However, the environment can be controlled in many applications. Systems that use multiple gray levels are not as dependant on the lighting conditions, but extra processing will be required to extract features from the image.

An additional consideration is the type of scene to be analysed. If the scene is two dimensional (2D) the task is simpler than is the case if the scene is three dimensional (3D). Some scenes, even though they are three dimensional, can be regarded as two dimensional, for example, satellite pictures of the earth. However, when a scene contains solid objects at a relatively close distance the 3D nature of the objects cannot be ignored. The brightness of surfaces depends on their orientation and lighting conditions. Additionally, some parts of the object may be occluded. Some of these problems can be alleviated by placing constraints on the system, for example, fixing the viewing angle. Ideally, though, objects should be recognisable from any viewpoint and under varying lighting conditions from a single image. To reduce the complexity of this process feature-based recognition can be used. Instead of working with the entire object simple objects are used. The complex object is considered as a sum of these simple object features. The simple objects should be chosen so that they are easy to detect in an image, that some of them will always be visible (independent of orientation), that they are distinctive and that they determine the position and orientation of the complex object.

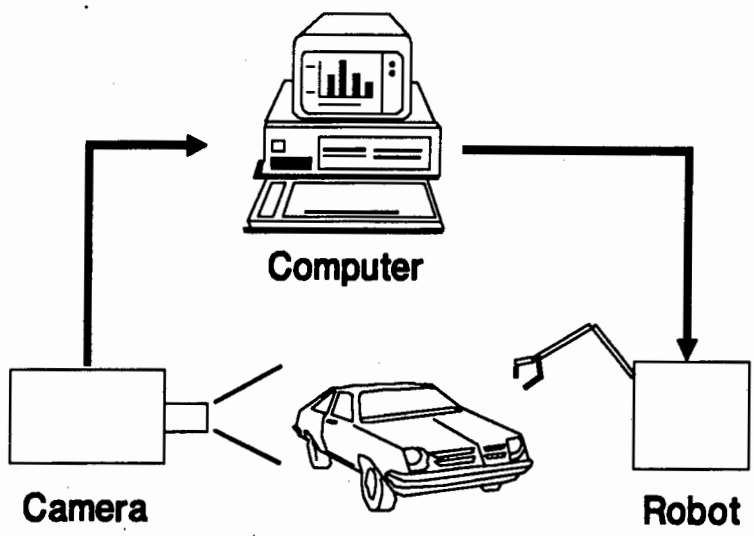


Figure 1.1. Typical hardware for a robot with machine vision. The capability to "see" makes the robot more versatile. It can be adapted easily to new tasks.

Therefore, the problem can be stated as: the recognition of a number of objects from a single image at unknown orientation using feature-based recognition. Once the primitive objects have been extracted from the scene geometric models can be used to recognise the complex objects. One of the most popular geometric models in vision research is polyhedra (Besl, 1988) in which case volumes are considered as being bounded by planar polygons. Other geometric models are: Quadric models in which objects are bounded by quadric surfaces such as spheres, cylinders or ellipsoids; Octrees in which volumes are subdivided into cube primitives; and Superellipsoids. Such geometric models are used in mechanical engineering to visualise mechanical part designs (Besl, 1988). They are also used in architecture and the film and television industry. Current research into these models is usually directed towards faster algorithms or more realistic displays. However, before the geometrical model can be applied, the primitive objects must be extracted.

Machine vision tasks usually divide naturally into these two levels, that of image-understanding (high level tasks) and image-analysis (low level). Image-understanding requires consideration of the entire image as well as reference to stored data. The low level

tasks do not require any previous knowledge, and typically only operate on a small area around each pixel. There is, however, often no clear distinction between the two levels. Essentially, the low level must sort the large amount of information presented to the system into a small, relevant set, which the high level then interprets. The low level must be fast and efficient, but at the same time provide the richest possible description of the scene. These low level tasks include segmentation, thresholding, edge detection, correlation, the Hough transform and many more.

Correlation requires that the set of all objects that might occur in a scene be known *a priori*. Additionally the scale and orientation of the objects must be fixed. The Hough transform (which was developed in 1962 by P. Hough as a means for recognizing straight lines in an image), on the other hand, can provide a description without such knowledge. This is one of the properties that makes this particular technique important in a flexible and 'intelligent' machine vision system. Other properties that make this method attractive are insensitivity to noise and to missing portions of a line. A further property is its inherent efficiency (the number of calculations is proportional to n, the number of image points, as opposed to n^2 for correlation). As a result the Hough transform has been applied to many different tasks, for example, to classify wooden boards into different quality classes (Poelzleitner, 1986) and to correct for linear variation of background illumination in images (Nixon, 1985).

The Hough transform operates on a binary, edge enhanced image. The input gray scale image is first subjected to an edge detection operator. The edge detection process eliminates a large amount of information and so fulfills one of the goals of a low level process, but, since all edge detectors are essentially differentiators, it also amplifies high frequency noise already present in the image. The edge operator may be one of many types, for example: Sobel, Roberts, Laplacian, or non-linear. A threshold is applied to the output of the edge detector to obtain a binary image. In the ideal case there are 1's where there are edges and 0's elsewhere. In the practical case there are also spurious 1's due to noise and the edges are not continuous. The Hough transform rejects most noise values and ignores small discontinuities in the edge.

The Hough transform is, therefore, an appropriate method for finding lines in an image. The information about line position can be used to extract polygons from the image. Since the lines originate from the edges of planes in the image, these polygons can be used as input to a geometric model for object recognition. This approach is an example of

"syntactic pattern recognition". Configurations of simple features are extracted from the image using rules about these configurations (Rosenfeld, 1988). Then configurations of these configurations are extracted and in this manner descriptions of complex objects can be built.

There is one difficulty when using the straight line Hough transform to extract line information from the image. That is, the Hough transform finds lines in the mathematical sense only. The lines are assumed to be infinitely long and infinitely narrow. Polygons are actually built up of line segments, so the positions of the ends of each segment are required. Operators exist that are specifically designed to detect corners, but they are usually computationally expensive (Gu and Huang, 1985; Wu and Rosenfeld, 1983). On reflection, it was anticipated that the end-point information could be extracted from the Hough transform directly.

The reasoning for this was as follows. The Hough transform has been shown to be a particular case of the Radon transform (Deans, 1981). The Radon transform is the process of taking projections at various angles around a solid body. It is the transform which arises in Computer Aided Tomography (CAT scan). More importantly, the Radon transform can be inverted. The image can be approximated by a reconstruction from the projections. It is this process that allows doctors to view slices through a patient. The inverse of the Radon transform can also be used to invert the Hough transform. Since the Hough transform is invertible (at least approximately) it follows that all of the information from the original image is present in the Hough transform representation.

This concept forms the basis for this thesis. The aim of the project was to understand the manner in which the Hough algorithm works, and with this understanding to determine how much information can be extracted using the Hough transform. The resulting parameter space, or Hough space, should be examined to determine how image features are represented. Additionally, the Hough transform should initially be implemented on a general purpose computer, but the viability of implementing it on dedicated hardware should be investigated. The layout of the thesis reflects these aims. Chapter 2 deals with the Hough transform and its variations. Chapter 3 covers the Radon transform, showing its relation to the Hough transform. It also describes the inverse transforms. The practical details of implementation are given in chapter 4. Chapter 5 describes methods for solving some image processing problems and gives results. Chapter 6 describes theoretical extensions for other types of Hough transform.

In addition, computer programs were required for the display of images and some of the basic image processing functions. Some of these were available on the VAX, but none were available for the IBM PCs. A suite of programs was written for the PC that enable the user to display images and perform some simple image processing, such as: Sobel edge detection, threshold, median filtering, low pass filtering, general 3x3 convolution, histogram display, histogram equalisation and more. A description of these programs appears in appendix A.

CHAPTER 2

2. The Hough Transform

A common problem encountered in image processing is the detection of collinear points in an image. This could be achieved by testing the lines formed by each pair of points, but this method requires computation in the order of n^2 for n points. The Hough transform, which was developed by Hough (1962) as a means for recognizing straight lines in an image, requires n.d (d is a constant related to angular resolution) computations and is therefore only linearly dependent on the number of points (Duda and Hart, 1972). Other pattern matching methods, such as correlation could be used, but the Hough transform has the advantage when the image is sparse (Rosenfeld, 1990).

Apart from being more efficient, the Hough transform is also less sensitive to noise and missing portions of the line or image to be detected. The Hough transform has also been extended to detect other features, such as circles, ellipses, hyperbolae, and even non-analytic shapes.

The input image to a Hough transform is an edge detected picture consisting of only the magnitude and direction of the edges. There are two extremes in the type of edge detector that can be used. At the one end are the edge detectors that give no information about edge direction. At the other are edge detectors that give the exact direction. In between are, obviously, edge detectors that estimate the direction to within a given accuracy. The more information about the edge direction available, the less the effort in Hough space, so effort in the edge detection can be traded off against effort in the Hough transform. Therefore, it is necessary to investigate the types of edge detection that can be done.

The first part of this chapter deals with edge detectors. In the second part the various types of Hough transforms are discussed, and finally the properties of the straight line version are examined in some detail.

2.1 Edge Detection in a Gray Level Image

Various operators can be used to detect edges in a gray level image. Most consist of a template (usually 3x3 pixels) which is convolved with the image in the spatial domain. This enhances the edges, which after thresholding, produces the binary edge detected picture required by the Hough transform.

Edges are in fact intensity discontinuities and usually represent the outline of objects. Human vision can identify many objects simply from their outlines. An example of the application of this idea is to transmit only the most important information in a picture, namely the edges. This was done by Pearson and Robinson (1985) when sending pictures for the deaf over a standard telephone line, which has a very limited bandwidth. The point is that the Hough transform is processing the most important information in a scene.

Most edge detectors compute the direction and magnitude of a discontinuity in an image. They are, however, also sensitive to noise as they are essentially differentiators. Many different types exist, such as gradient, Laplacian or template matching. The performance of the different types varies, depending on the conditions in the image. Edges may be step, ramp or curved.

Three types of edge detectors will be considered here. Firstly, linear edge detectors that are implemented by convolution in the space domain. Among these are the well known Roberts, Sobel and Laplacian operators. Secondly, linear edge detectors that are implemented in the frequency domain because of their large mask size. Thirdly, non-linear space domain operators will be considered.

2.1.1 Linear space domain operators

This type of edge detector is usually implemented as several 3x3 masks which are convolved with the input image. The outputs from the masks are combined to give a magnitude and direction for the edge.

Edge detectors estimate the gradient of the image at a point. If the image is denoted by $f(x,y)$ and the slopes in two orthogonal directions are $\delta f/\delta x$ and $\delta f/\delta y$, then the slope in any direction, θ, is given by (Rosenfeld and Kak, 1982):

$$\frac{\delta f}{\delta x'} = \frac{\delta f}{\delta x} \cos\theta + \frac{\delta f}{\delta y} \sin\theta$$

and the direction of greatest magnitude is:

and the direction of greatest magnitude is

$$direction = \arctan \frac{\delta f/\delta y}{\delta f/\delta x}$$

$$magnitude = \sqrt{((\delta f/\delta x)^2 + (\delta f/\delta y)^2)}$$

In a digital image the first derivative, $\delta f/\delta x$, is found by taking the difference between adjacent pixel values. An operator based on the first derivative is the Roberts operator. This convolves the two masks:

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

with the image. However, diagonal edges are biased against because they are weighted with $\sqrt{2}$ as opposed to 2 for vertical or horizontal edges (Rosenfeld and Kak, 1982). Also, the small mask size makes this operator very sensitive to noise.

The Laplacian operator, $\delta^2 f/\delta x^2 + \delta^2 f/\delta y^2$, uses the second derivative of the image. Converting these to differences gives the convolution mask:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

For positive only responses to edges the absolute value of the output can be taken. However, the Laplacian operator tends to be very sensitive to noise (figure 2.1(a)).

To reduce the effects of noise, the image can be smoothed first. Such operators are called 'difference of averages'. An operator based on the weighted difference of averages is the Sobel operator (figure 2.1(b)).

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

(a) (b)

Figure 2.1. Examples of convolution edge detectors. (a) Laplacian edge detection. (b) Sobel edge detection.

The magnitude of the edge is the greater of the two outputs. Although this operator does not have an ideal response, in that its response to diagonal edges is weaker than that to vertical or horizontal edges, it can be implemented with very few calculations, most of which are additions or subtractions.

The outputs of each of the above edge detectors must be threholded at some level to create a binary image suitable for the Hough transform.

2.1.2 Linear frequency domain operators

A linear frequency domain operator would be implemented as a two dimensional Fourier transform followed by a multiplication by a mask followed by an inverse 2D Fourier transform.

The advantage of this method is that the number of computations required is independent of the mask size. Therefore, much larger masks can be used giving improved noise performance. The disadvantages are the large storage requirements for the mask and Fourier transform results.

A particular example of this is homomorphic filtering. Homomorphic filtering can be implemented as a convolution mask, but since it works by subtracting the local average from the pixel value, it needs a large area of support for a reliable average. Therefore it is more efficient to implement this type of filter in the frequency domain.

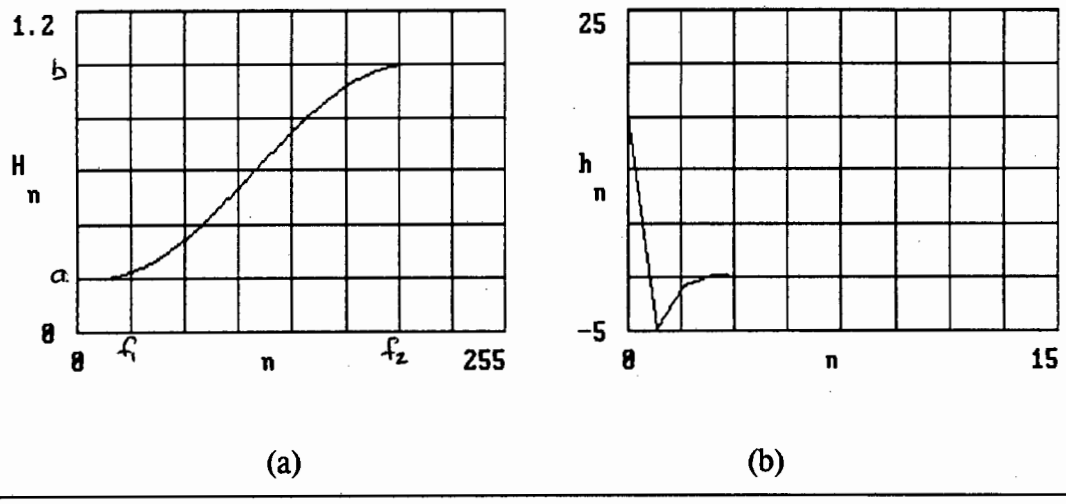


Figure 2.2. The response of a homomorphic filter used for edge detection. (a) Frequency response. (b) Time response.

The frequency response of a homomorphic filter is shown in figure 2.2. The parameters f1, f2, a and b can be adjusted to give varying degrees of ringing and low frequency suppression. This type of filter is used for contrast enhancement, but by setting appropriate values for the parameters, it can be used as an effective edge detector.

2.1.3 Non-linear space domain operators

The valley operator suggested by Pearson and Robinson is a non-linear edge operator. It first performs a rough detection, and if this process reports an edge a more detailed operation is performed. For the initial estimate a 3x3 mask is used so the process is fast. If this reports a likely edge, a 5x5 mask is used to accurately determine whether an edge is

actually present. This method also has the capability of estimating the edge direction to within 1 out of 8 possible directions.

The method used here is a modified version of a detector suggested by Pearson and Robinson. This edge detector was found to be fairly insensitive to noise, fast and to generate directional information as well as magnitude. Directional information is used in some versions of the Hough Transform. A set of oriented valley operators at various orientations is applied to the image. The oriented valley operator was chosen over the isotropic version due to its decreased sensitivity to noise. The detector is a 5x5 mask which gives improved noise performance over a 3x3 mask. The computations required are minimised by performing logical decisions, first on a 3x3 mask, then if an edge is likely then the full 5x5 mask is applied.

Figure 2.3. Non linear edge detection.

Figure 2.3. shows the result of applying a set of logical operators (one for each of eight directions). An example of one of these operators, the operator to detect a vertical valley, is shown below:

a	b	c	d	e
f	g	h	i	j
k	l	m	n	o
p	q	r	s	t
u	v	w	x	y

```
if ((l-m)>T1 or (n-m)>T1)
then if (f+k+p+j+o+t-2(h+m+r))>T2
and (g+l+q+i+n+s2(h+m+r)) >
    (f+k+p+h+m+r-2(g+l+q))
and (g+l+q+i+n+s-2(h+m+r)) >
    (h+m+r+j+o+t-2(i+n+s))
      then there is a valley
          through m.
where a-y are pixel elements and
  T1,T2 are thresholds.
```

2.2 Different Implementations of the algorithm

2.2.1 Straight line detection

Any straight line can be described in terms of two parameters. For example a line, $y = m \cdot x + c$, is uniquely described by m and c (see figure 2.4). Hough suggested forming an array of accumulators for m and c and then performing then following algorithm:

```
for all (x,y) edge points in image
  for all values of c
    calculate m=(y-c)/x
    increment accumulator array A(m,c)
```

The accumulator now consists of a series of peaks. These peaks correspond the lines in the image. A peak at a point m, c means that there is a line in the image with equation $y = mx+c$. The relative location of these peaks conveys information about the contents of the image.

In this implementation c can be limited to a specified range, but m is unbounded. As the size of the accumulator array is bounded this poses a problem. This can be overcome by performing two orthogonal transformations. The first transform is applied directly to the image to be transformed; the second is applied to a version of the image that has been rotated by 90 degrees. Thus lines that are near vertical for one transform are near horizontal for the other (Rosenfeld, 1969).

An alternative to this 'Twin Hough Space' technique is to use another parameterisation of the line, the ρ,Θ parameters (see figure 2.4).

The algorithm in this case is:

```
for each image point (x,y) where there is an edge
  for all Θ
    calculate ρ = x·cos Θ + y·sin Θ
    increment A(ρ,Θ)
```

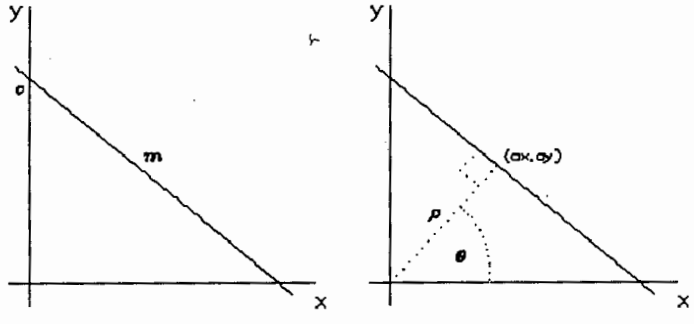
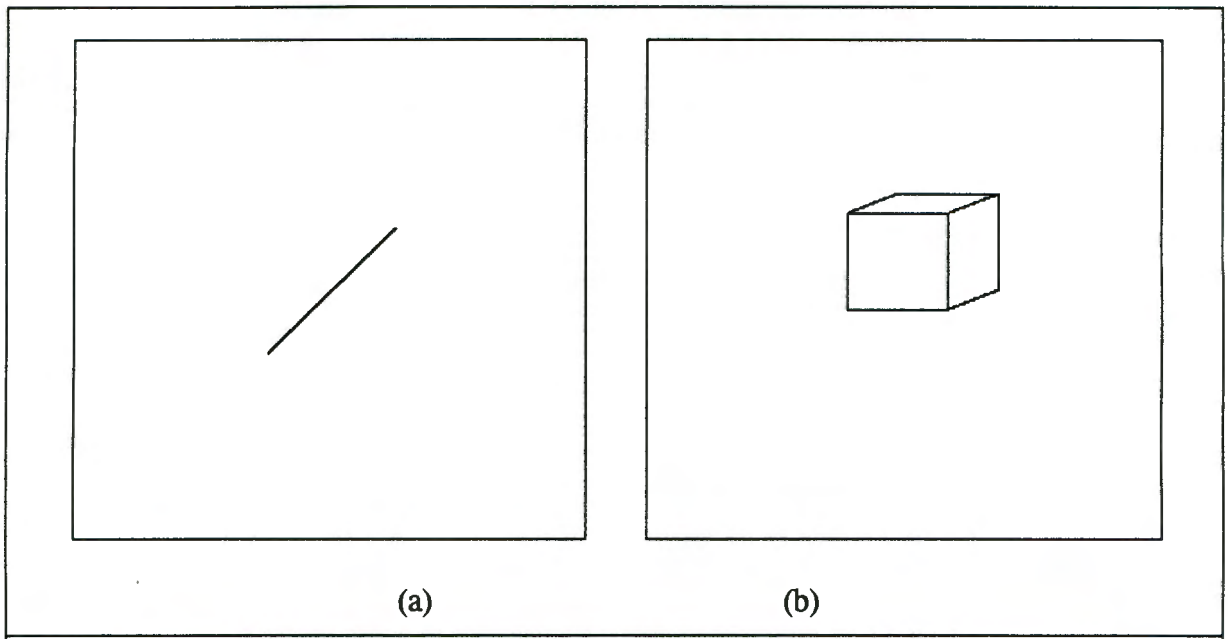


Figure 2.4. Two parameterisations of a straight line.

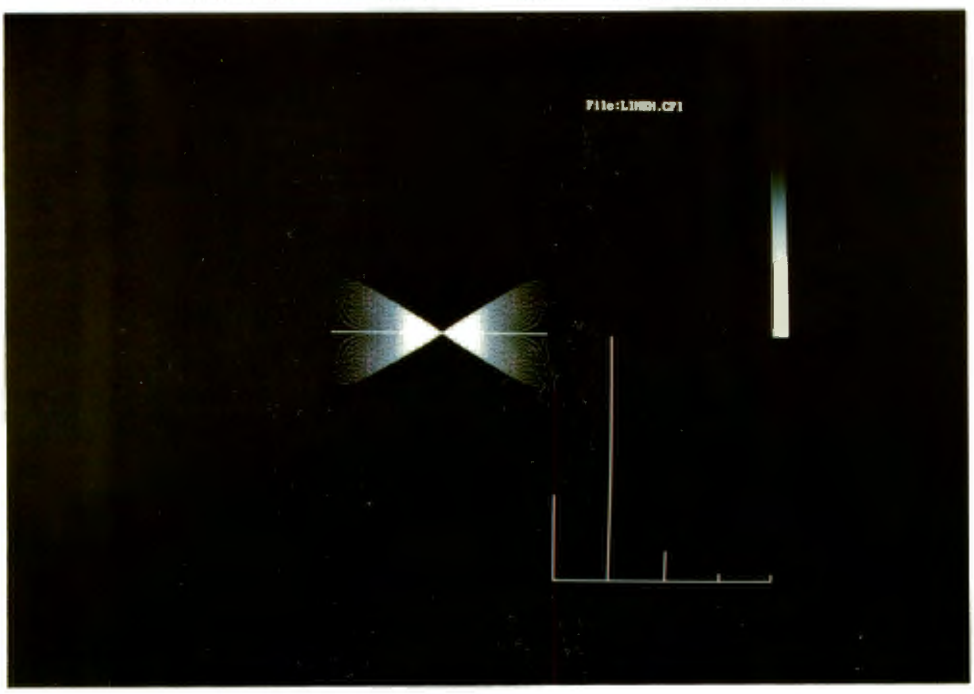
For every edge point in the image a sinusoid is plotted in the Hough space (the accumulator array). Where many of these sinusoids intersect a peak is formed. The sinusoids generated by a set of collinear points all intersect at a unique position. Each peak in the accumulator array (at $A(\rho,\Theta)$) corresponds to a line with parameters ρ and Θ in the image (Duda and Hart, 1972). Figure 2.5 shows examples of the ρ,Θ Hough transform on some simple images.

In this case ρ can be limited to a maximum value R. This will detect all straight lines within a retina radius R in the image. Theta is confined to the range $(0,\pi)$ and ρ may be positive or negative. Thus both parameters are bounded and therefore the accumulator array may be of finite size. The size of the array determines the quantization of ρ and Θ. If the range for Θ is divided into d segments, the computational requirement for the Hough transform is n.d (as before). The resolution of ρ has no effect on the efficiency. The resolution of both parameters affects the accuracy of the transform. If the resolution is too coarse some lines will not be detected. If the resolution is too fine, one line will not

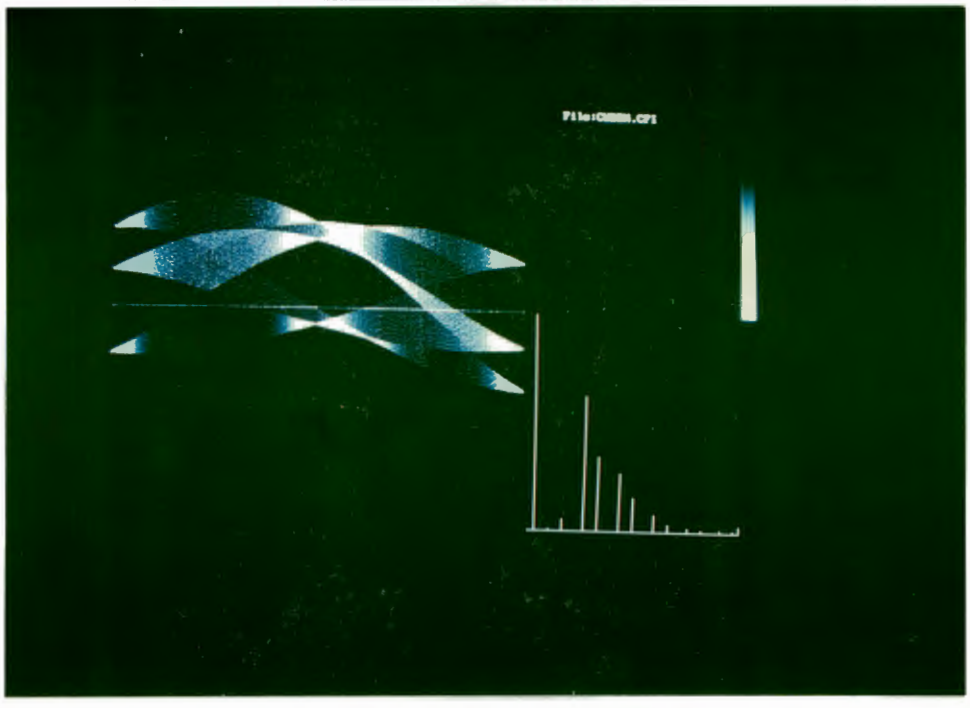
generate a peak, but a cluster of points. The inverse of the Hough transform can be found by drawing the line corresponding to each peak.



(a) (b)



(c)



(d)

Figure 2.5. Images and their Hough transforms. (a) Straight line. (b) Cube. (c) Hough transform of straight line. (d) Hough transform of cube. The horizantal axis of the Hough transform is the angle, θ, and the vertical axis is ρ

2.2.2 Curve Detection

The straight line Hough transform can be extended to detect any conic section. This includes circles, ellipses, hyperbolae and parabolas. These curves are of the form $\mathbf{f(x,a)=0}$ where $\mathbf{x}$ is an image point and $\mathbf{a}$ is a parameter vector. This method was suggested by Ballard (1981).

The algorithm is then:

```
form an accumulator array A with all elements set to 0
for each edge pixel x
  compute all a such that f(x,a)=0
  increment A(a)
```

The order of the efficiency of this algorithm is $N.M^{m-1}$ where N is the number of edge pixels, m is the number of parameters and M is the number of values each parameter can take. If an edge detector that generates the edge direction as well as magnitude is used, the efficiency of this algorithm can be improved.

If the equation of the curve is differentiated:

$$df/dx = \tan \left[\phi(\mathbf{x}) - \pi/2 \right]$$

where $\phi(\mathbf{x})$ is the gradient direction as given by the edge detector.

The algorithm is now given as :

```
form an accumulator array A with all elements set to 0
for each edge pixel x
  compute all a such that f(x,a)=0 and df/dx (x,a)=0
  increment A(a)
```

This improves the efficiency to $N.M^{m-2}$. Thus for a circle, which has three parameters, the effort will be proportional to N.M as opposed to $N.M^2$. Using gradient information from the edge detector can reduce this (Kimme, 1975).

Conic sections, apart from the circle, are dependant on orientation. Arbitrary rotation angles can be accounted for by introducing a further parameter, θ:

$$df/dx = \tan \left[\phi(\mathbf{x}) - \theta - \pi/2 \right]$$

2.2.3 Non analytic shape detection

Non analytic shapes can be detected using the Hough transform in its generalized form (Ballard, 1981). This method matches a given template of the shape to the image and produces a peak in the accumulator array if the shape is found. Analytic shapes can also be found with this method, so it includes the previous classes of lines and curves.

The following parameters are defined for an arbitrary shape:

$\mathbf{a} = \{\mathbf{y}, \mathbf{s}, \Theta\}$

where: $\mathbf{y} = (x_r, y_r)$ is a reference origin for the shape

$\mathbf{s} = (s_x, s_y)$ is a pair of orthogonal scale factors

Θ is the orientation

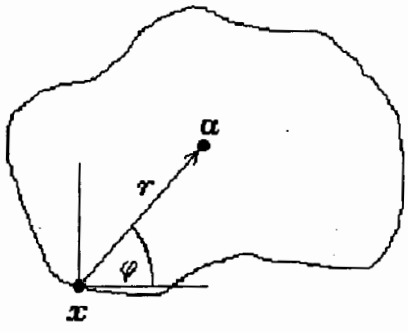


Figure 2.6. Geometry for Generalized Hough Transform

For each point $\mathbf{x}$ on the boundary of an arbitrary shape, there is a gradient direction ϕ. Now the point $\mathbf{a} = \mathbf{x} + \mathbf{r}$ in the accumulator array is incremented. See figure 2.6 for the diagram.

Since the values of $\mathbf{r}$ are not described by any analytic function the values are stored in a table called the R-table. The R-table is constructed from the following algorithm:

```
choose a reference point y for the shape
for each boundary point x
  compute φ(x) and r = y - x
  store r as a function of φ
```

In general, there may be multiple values of $\mathbf{r}$ for each ϕ.

The transform presented this far is dependant on rotation and scaling of the image. In a machine vision system it is often desirable to detect the presence of an object independent of its scale or orientation. Two extra parameters can be added to the shape description to allow this. The accumulator array now consists of five dimensions $(\mathbf{y}, s, \Theta)$.

A change in scale by some scale factor, s, can be represented by a transformation T_s.

$$T_s[R(\phi)] = sR(\phi)$$

i.e all the vectors are scaled by s.

Similarly a rotation by θ can be represented by T_θ where:

$$T_\theta[R(\phi)] = \text{Rot } \{ R[(\phi-\theta) \bmod 2\pi], \theta\}$$

i.e. the table and the vectors are rotated by θ.

2.2.4 Ellipse detection

Ellipses are members of the conic section family of curves described in section 2.2.2. However, ellipses are important in machine vision applications. For this reason they will be discussed in further detail in this section.

Ellipses arise frequently because any circular object, viewed from an angle, becomes an ellipse. In many industrial systems parameters, such as diameter and position, of circles need to be determined.

The Hough transform of an ellipse can in principle be found in the manner described in the previous section. However, an ellipse has five parameters, for example: lengths of semi-major and semi-minor axes, x,y coordinates of centre and a rotation angle. This means that the Hough space would be five dimensional, requiring extremely large storage and many computations. To reduce the dimensionality of the problem several methods have been proposed.

One approach is to decompose the problem into a number of sequential stages. A three stage method was proposed by Muammar and Nixon (1990) (each stage being two dimensional). This method is a modified version of that proposed by Tsuji (Tsuji and Matsumoto, 1978). The idea is to break the problem down into more easily solvable stages. The first stage is to estimate the centre of the ellipse by considering the tangents at two points on the ellipse. In some cases this is enough to identify the position of the ellipse

(Ruther, 1990). The second stage is the determination of ellipse orientation, and the third stage the estimation of the major and minor radii.

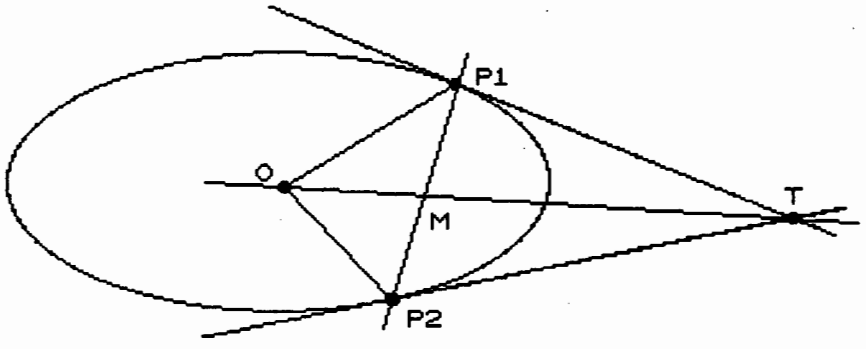


Figure 2.7. Ellipse showing the points M and T used to calculate the ellipse centre.

The first of these stages, finding the centre, is based on the following property. For two points $P_1(x_1,y_1)$ and $P_2(x_2,y_2)$ lying on the boundary of an ellipse with tangents at P_1 and P_2 that intersect (see figure 2.7). If a line is drawn between the point of intersection, $T(t_x,t_y)$, and the midpoint of P_1 and P_2, $M(m_x,m_y)$, it will pass through the centre of the ellipse. The coordinates of T and M can be found using geometry, so the accumulator can be incremented along the line TM. If the approximate size of the ellipse is known then the accumulator can be incremented only for relevant portions of the line TM. Similarly, point pairs, whose separation is small, should be excluded because the estimation of the gradient of the tangents is subject to large error. When all relevant point pairs have been considered, a peak will be formed in the accumulator array at the centre of the ellipse. Errors caused by any irregularities in the shape of the ellipse will cause background noise in the accumulator array or spurious peaks located away from the main peak, but will not affect the position of the peak. With least mean squares methods, such errors affect the calculation of the centre (Ruther, 1990). A disadvantage of the Hough

method is that the number of computations required is $O(N^2)$, where N is the number of points on the ellipse (compared to $O(N\log N)$ for the straight line version).

The second stage is to estimate the ellipse orientation. This uses the fact that an ellipse has two lines of symmetry, both of which pass through the centre (which is now known). The procedure is to determine those midpoints, M, and tangent intersections, T, which lie on either axis of symmetry. This will be the case if the distances from the origin to P_1 and P_2, OP_1 and OP_2 are equal. The ellipse orientation is then the angle subtended by OM or OT. By averaging the contribution from all such angles, the orientation can be estimated.

The third stage is to estimate the major and minor radii (a and b, respectively). To find these, the ellipse is rotated about the estimated centre, by the estimated angle of orientation. The radii, a and b, can be calculated by substituting pairs of points into the standard equation for an ellipse.

A second approach is to reduce the number of dimensions by accumulating all the votes for all possible orientations in a single plane in parameter space (Davies, 1989). This method uses the generalised Hough transform which was discussed in the previous section. The essence of this method is that a (r,ϕ) table is built for the locus of all possible ellipse centres for ellipses of different orientations. Ellipses of different sizes are catered for by scaling the values in the table.

2.3 Peak finding in Hough Space

Once the Hough space has been calculated, it is necessary to find the position of the peaks. The peaks may be found to a high degree of accuracy, or simply to pixel accuracy. Also, the characteristics of the peak depend on the resolution of the Hough transform. If the Hough transform is calculated at a few angles, then several lines may have the same peak. If on the other hand a large number of angles are used, the peak may be spread over more than one pixel and become a cluster. In the images and transforms used here the latter case is more relevant, because all the Hough transforms used have 512 discrete angles between 0 and π.

The conceptually simplest method is to threshold the resulting Hough space at some manually determined level. Unfortunately the height of a peak depends on the size of the

object that it represents. To a lesser degree it also depends on the position of the object. Therefore it is very difficult to automatically determine a threshold, and in most cases it is set manually. If however, all of the objects of interest are the same size, and this size is known, then a manual threshold will work reasonably well. This is borne out by experimental results. If these conditions are not met, other methods can be used.

One alternative is to use the converging squares algorithm. This divides the image into smaller and smaller squares, always moving towards the region of highest density (O'Gorman, 1984). This algorithm can be applied iteratively to find all of the peaks in order of density. Once an area has been found to be a peak it is excluded from the next search. The search can be stopped once the highest density falls below some preset level.

A third method is an iterative method. The first step is to find the highest global peak. Assuming that this corresponds to an object in the image, decrement the accumulator array at all points that would have been incremented by such an object. Now repeat the process until all peaks have been found.

Apart from being computationally intensive, this method has one major flaw. In the case of a straight line, for example, the peak position only indicates the ρ and θ parameters. The line is assumed to be infinitely long (using 'infinite' in the loose sense that the line extends beyond the borders of the image). So, decrementing the accumulator array for an infinitely long line, will detract from the height of other peaks which should not have been affected, leading to the possible elimination of small peaks. The solution to this problem is to find the start and end points of each line.

2.4 Properties of the Straight line transform

For the remainder of the thesis, the discussion will be limited to the ρ,θ straight line version of the Hough transform. This version is easy to implement on a computer as there are only two parameters. Since the computations and storage space required are related exponentially to the number of parameters, it is not practical to implement some of the other versions without modification. However, generalisations will be made to show how the methods applied to the ρ,θ version can be used with other parameterisations.

2.4.1 Inverse of the straight line Hough Transform

Using the ρ,Θ parameter transform, each peak gives the ρ and Θ necessary to characterise a line. The gradient-intercept version can then be calculated from this.

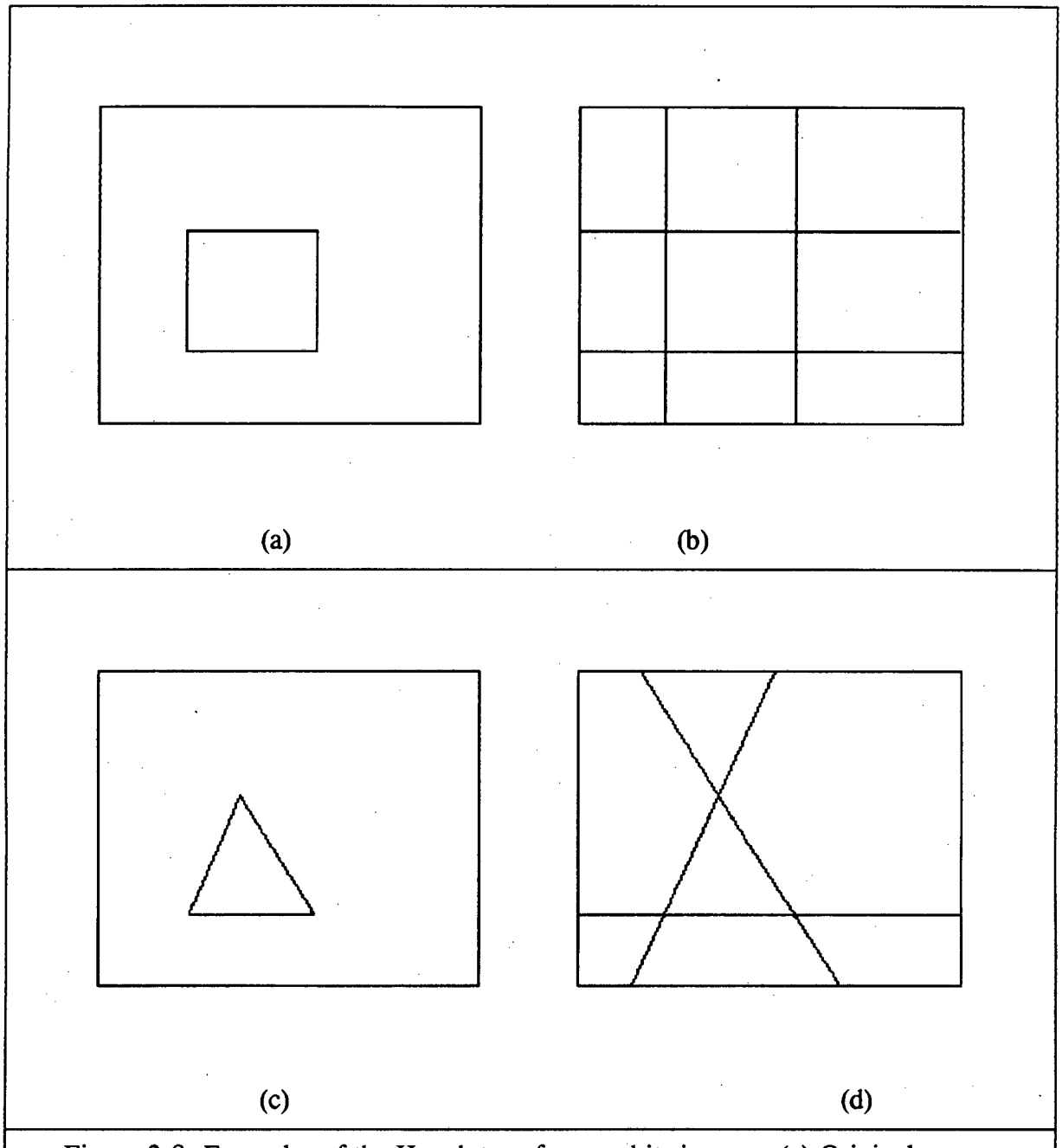


Figure 2.8. Examples of the Hough transform and its inverse. (a) Original square (b) Square reconstructed using information from Hough transform (c) Original triangle (d) Reconstructed triangle

The general form is $y = mx+c$, m is known from $m = -1/\tan \theta$ and c can be calculated because a point on the line is known (ax,ay) (see figure 2.4).

$$ax = \rho \cdot \cos \theta$$
$$ay = \rho \cdot \sin \theta$$
$$c = ay - m \cdot ax$$

If the line is vertical, or near vertical, m will be large. In this case it is more convenient to use the form $x = ny+d$. Here:

$$n = \tan \theta$$
$$d = ax - n.ay$$

The values for m and n can be calculated first, then compared and used to decide which form to plot.

Implementation of this method maps a series of points in Hough space back to an image. Examples of this mapping are given in figure 2.8.

2.4.2 Relative Positions of Peaks

Several conclusions about the image can be drawn from the position of peaks in the Hough space (Wahl and Biland, 1986). The Hough space is a plot of ρ, on the vertical axis, against θ, on the horizontal axis.

1. Vertically aligned peaks in Hough space correspond to lines in the image with the same slope (i.e. the ρ values may be different, but the θ values are the same).
2. The number of edges in the image corresponds to the number of peaks in Hough space. Collinear edges result in overlapping peaks in Hough space.
3. Lines which intersect have peaks that lie along a curve $\rho = x \cdot \cos \theta + y \cdot \sin \theta$.

These properties can be used to describe three dimensional polyhedra on the basis of their two-dimensional image representation.

2.4.3 Vertex Finding

The Hough transform is an algorithm that extracts features from an image. In this case the features extracted are straight lines. To identify objects in the scene it is necessary to generate a description of the scene. In CAD systems 3 dimensional objects are modelled using coordinate triples of the vertices. The information about the lines in an image (from the Hough transform) can be used to calculate the position of vertices.

The Hough transform maps an image from (x,y) coordinates to a (ρ,Θ) plane. Each point in the image is mapped to a sinusoidal curve in the Hough transform space. The equation for each of these curves is (Duda and Hart, 1972):

$$\rho = x \cdot \cos\Theta + y \cdot \sin\Theta$$

Collinear points in the image will map to curves that all intersect at a point in the ρ,Θ plane. The ρ and Θ position of the peak give the distance along the normal to the line to the origin, and the angle between the normal and the x-axis.

In this manner the equation describing any line in the image can be found. It should be noted that the Hough transform does not preserve the segment information (there is no information about where the line starts or ends). Each line is assumed to extend beyond the borders of the image.

However, given the ρ and Θ parameters of two or more lines, the intersections can be calculated. Assume that two such lines are denoted by ρ_1,Θ_1 and ρ_2,Θ_2. The x,y coordinates in the image plane of the intersection can be calculated as follows:

First the ρ,Θ parameters are converted to the form $y = m.x + c$ using:

$$m = -\frac{\cos\Theta}{\sin\Theta} \quad , \quad c = \frac{\rho}{\sin\Theta}$$

Then :

$$x_i = \frac{c1 - c2}{m2 - m1} \quad , \quad y_i = m_1 . x_i + c_1$$

In the practical case, checks are also performed for vertical lines (which cause the gradient to be infinite) and for parallel lines (which do not have an intersection).

The algorithm for finding all intersections would then be:

```
for i = 1 to all lines
  for j = i+1 to all lines
    calculate intersection of lines i,j
```

However, a problem is shown in figure 2.9.

This problem of false vertices causes difficulties when classifying the image into object primitives (e.g. pyramids or cubes) (Wahl and Biland, 1986). If the segment information about the line can be kept then the false vertices can be rejected. The following proposed method will do this.

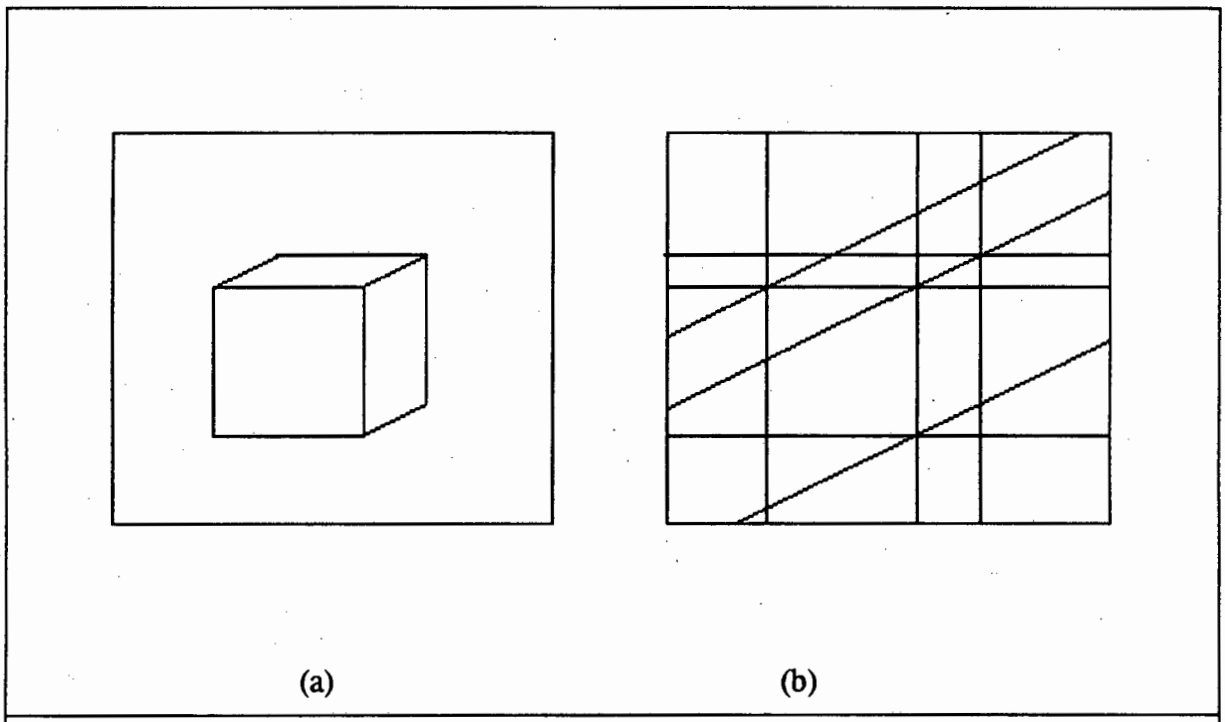


Figure 2.9. False intersections due to lines being infinitely long. (a) Original cube. (b) Cube reconstructed from transform. Note that there are line intersections that do not correspond to vertices in the original image.

2.4.4 Algorithm for Finding Start and End Points

If a Hough transform is being used to detect lines in an image, then extra information, besides the ρ and θ parameters, can be extracted, without having to reprocess the entire image. The algorithm presented here will calculate the start and end points of each line segment.

First it is necessary to examine the workings of the Hough transform from a geometrical viewpoint.

Figure 2.10 shows the ρ,θ parameters of a line. Using geometry a conversion between these and the m,c parameters can be found. The slope of the normal can be calculated from $m = dy/dx$:

$$m_n = \frac{AB}{OB} = \frac{\rho \cdot \sin\theta}{\rho \cdot \cos\theta} = \frac{\sin\theta}{\cos\theta}$$

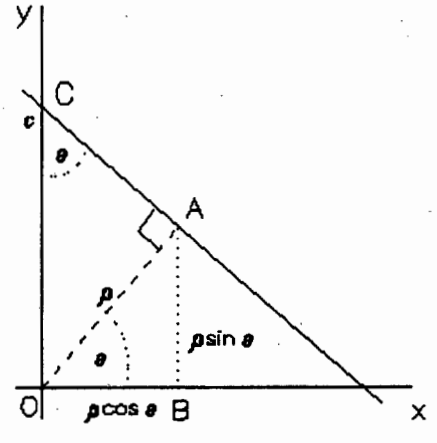


Figure 2.10. Conversion from ρ,θ parameters of a line to the $y = m \cdot x+c$ form.

Since the gradients of two orthogonal lines are related by the expression $m_1 = -1/m_2$, the gradient of the line is:

$$m = -\frac{\cos\theta}{\sin\theta}$$

By considering the triangle OAC, the value for c can be calculated. Angle AOC = $90 - \theta$ so angle OCA = θ. Using the sine rule:

$$\frac{OA}{\sin AOC} = \frac{OC}{\sin OAC}$$

$$\frac{\rho}{\sin\theta} = \frac{c}{\sin 90^{o}} = c$$

Now to consider what the Hough transform does. Assume that there is a line segment in the image denoted by $y = n.x+d$. This line segment starts at a point (x_1, y_1) and ends at (x_2, y_2).

The Hough transform algorithm is:

Take a point (x,y) on the line segment
For all θ from 0 to 180^{o}
  Calculate $\rho = x \cdot \cos\theta + y \cdot \sin\theta$
    and plot the point in ρ,θ coordinates

What this actually does is: for every discrete θ (which fixes the direction of the normal), calculate what ρ is needed to describe a line ($y = m.x+c$) that passes through the point (x,y). This can be proved as follows:

$$m = -\frac{\cos\theta}{\sin\theta} \quad \text{(as calculated above)}$$

$$c = \frac{\rho}{\sin\theta}$$

Therefore the equation of the line is given by:

$$y = -\frac{\cos\theta}{\sin\theta} \cdot x + \frac{\rho}{\sin\theta}$$

Rearranging this gives:

$$\rho = x \cdot \cos\theta + y \cdot \sin\theta$$

So, calculating this actually calculates the ρ parameter, for a given θ, of a line passing through (x,y). As θ is incremented from 0 to 180^{o} many such lines are found and their ρ,θ values are plotted in the Hough space to give a sinusoidal curve. When the process is repeated for a second point, there will be a certain line (with parameters ρ,θ) that passes through this new point as well as the old. The curve plotted by the new point will cross over the curve from the old point at this particular ρ and θ.

The whole process is repeated for each (x,y) point on the line. This results in a whole family of curves in the ρ,θ space (see figure 2.11(b)). This explains why a peak is generated when there are collinear points in the image.

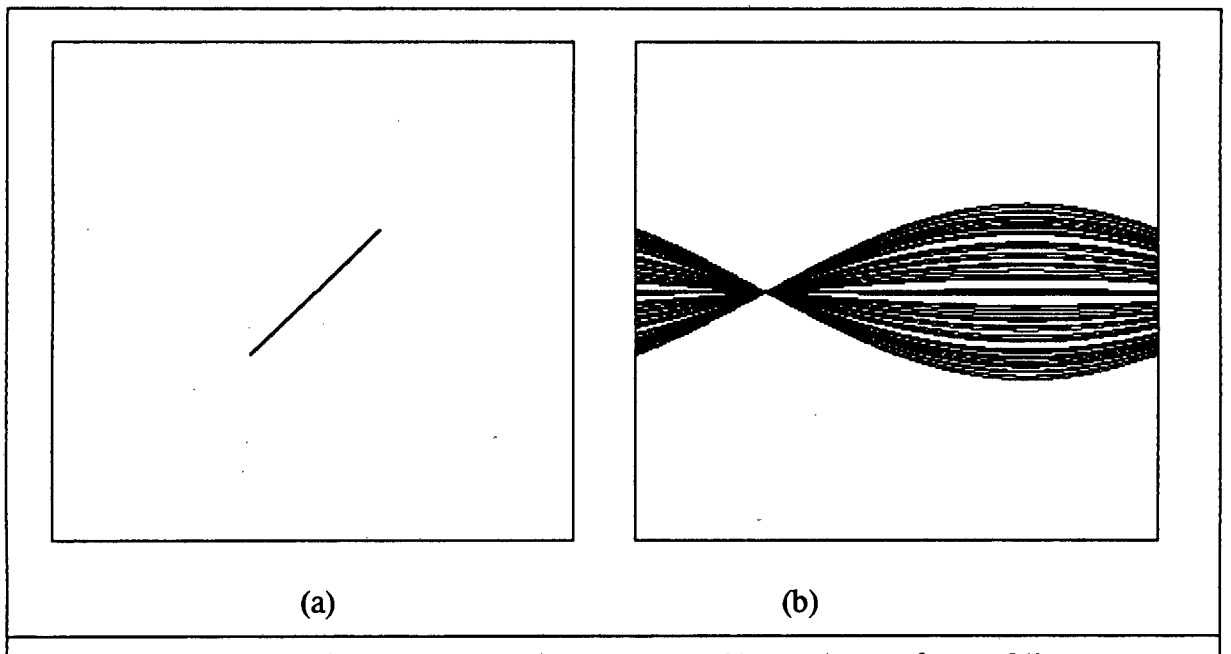


Figure 2.11. Converging curves. (a)Line segment. (b)Hough transform of line segment showing converging curves.

To view the situation slightly differently, fix θ at some value. Now $\cos\theta$ and $\sin\theta$ are constants. Call them a and b. Now:

$$\rho = x \cdot a + y \cdot b$$

Because all x,y points fall on the line segment in the image:

$$y = n \cdot x + d$$

This can be substituted into the above to give:

$$\rho = x \cdot a + (n \cdot x + d) \cdot b$$
$$= x\ (a + b \cdot n) + b \cdot d$$

which is a straight line. That is for a fixed θ, ρ is linearly related to x (and since x is linearly related to y, ρ is also linearly related to y). The term $(a+b \cdot n)$ corresponds to the slope. If this is positive, then the maximum ρ will correspond to the maximum x (which must be at one end of the image line segment). Similarly the minimum ρ will correspond to the minimum x (which must be at the other end of the line segment). If the slope is negative then the maximum ρ corresponds to the minimum x and vice versa. The important fact is that the extreme values of ρ correspond to the ends of the line segment.

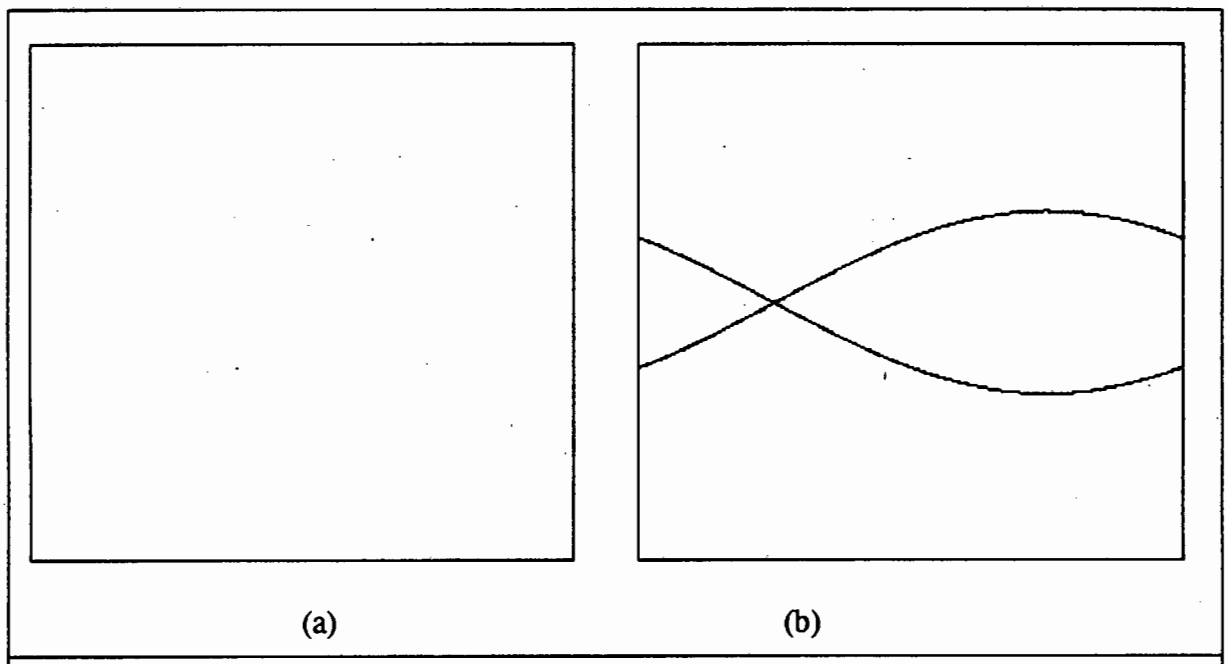


Figure 2.12. Hough transform of the ends of the line. (a)End points of line segment in figure 2.11. (b)Hough transform of (a). Note that this is the same as the extreme curves in figure 2.11(b).

The significance of this is seen in the Hough transform space. The curves plotted from each point on a line segment in the image plane converge to a point. Figure 2.11(b) shows many such curves generated from the line segment shown in 11(a) converging on a point.

The coordinates of the point give the ρ and θ parameters of the line segment. The outside curves of the 'bundle' are generated by the extremes of the line segment. This is also shown by example in figure 2.12, by taking the Hough transform of only the end points of the segment.

Each of these curves has the form:

$$\rho = x_e \cdot \cos\theta + y_e \cdot \sin\theta$$

where (x_e, y_e) is the coordinate of one end of the line segment. If ρ and θ can be estimated then x_e and y_e can be calculated. In order to do this, a small region around the convergent point is considered. Within this region the curves are approximated by straight lines. Two points on each of the extreme curves are found, and then (x, y) can be found for each end of the line. To find one end, consider two points on one 'curve' denoted by (ρ_1, θ_1) and (ρ_2, θ_2). Therefore:

$$\rho_1 = x \cdot \cos\theta_1 + y \cdot \sin\theta_1$$
$$\rho_2 = x \cdot \cos\theta_2 + y \cdot \sin\theta_2$$

Solving these two simultaneous equations for x and y gives:

$$x = \frac{\rho_2 \cdot \sin\theta_1 \cdot \sin\theta_2 - \rho_1}{\cos\theta_2 \cdot \sin\theta_2 \cdot \sin\theta_1 - \cos\theta_1}$$

$$y = \frac{\rho_1 - x \cdot \cos\theta_1}{\sin\theta_1}$$

This calculation is performed for each end of each line segment. When θ_1 is equal to 0^o or 180^o, $\sin\theta_1$ is equal to zero and y becomes indeterminate. In this case the algorithm fails.

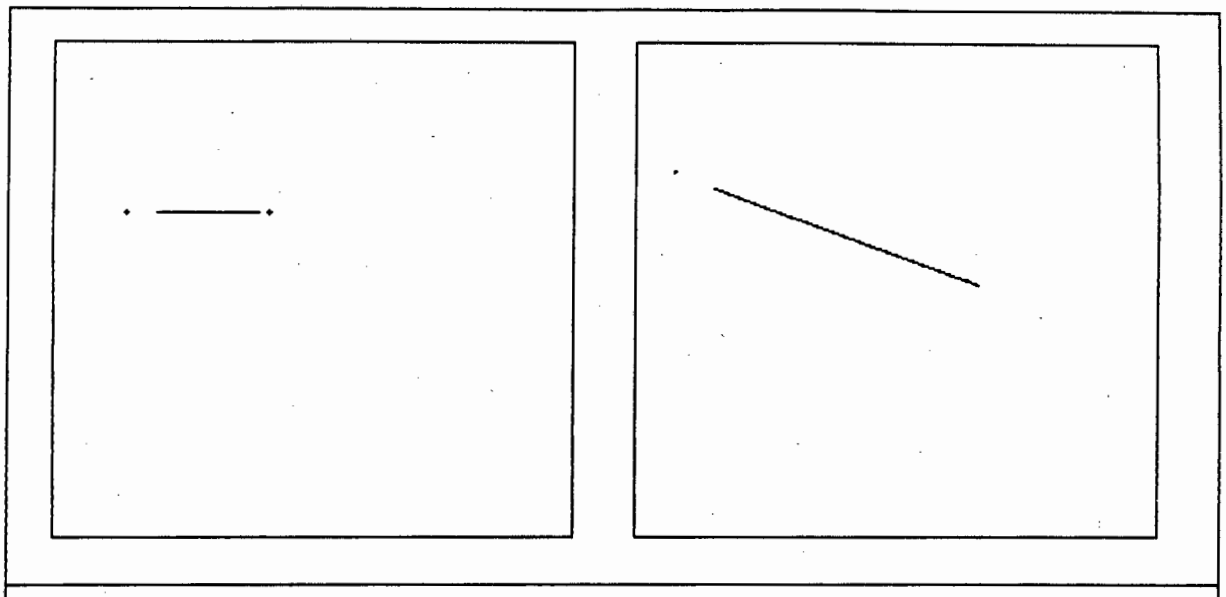


Figure 2.13. Images showing original lines and calculated end points. The small crosses show these points. The end points are calculated from the shape of the butterfly feature in Hough space

The above figures (figure 2.13) show the results of this algorithm on several line segments. Although not completely accurate, they show that the points can be calculated. Additionally, the algorithm may fail if there are multiple line segments in the image. The next chapter looks at means for inverting the Hough transform completely. This will allow for a more accurate method of calculating end points even in the presence of multiple line segments.

CHAPTER 3

3. Inversion of the Radon Transform

The Hough transform has been shown to be a special case of the Radon transform (Deans, 1981). In fact, the Radon transform itself has been used to detect linear features (Murphy, 1986). The Radon transform is important as it arises in many fields, most particularly in Computer Aided Tomography. In this application the Radon transform must be inverted to produce cross-sectional images of a patient. Techniques have been developed for efficiently computing this inverse and, due to the fact that the Hough transform is a special case, these techniques can be used to invert the Hough transform with no modification.

The first part of this chapter deals with the relation between the Hough and Radon transforms. The second part deals with inversion of the Radon transform. Two different methods are discussed and the relative strengths and weaknesses of each algorithm are shown.

3.1 The Radon Transform

The two dimensional Radon transform, as first studied by Johann Radon (1917), is given by the following integral:

$$R\{f\} = \int_L f(x,y)\, ds$$

where the function $f(x,y)$ exists in a domain D in the space R^2 and L is a line in the plane. ds is an increment along L.

If the equation of the line L is given in normal form, with ϵ being a unit vector in direction θ:

$$\rho = \epsilon \mathbf{x} = x \cdot \cos\theta + y \cdot \sin\theta$$

Then (Deans, 1983):

$$R\{f\} = \int f(\mathbf{x}) \cdot \delta(\rho - \epsilon\mathbf{x})\, d\mathbf{x}$$

$$= \int f(x,y) \cdot \delta(\rho - x\cdot\cos\theta - y\cdot\sin\theta)\, d\mathbf{x}$$

Which is the expression for the Radon transform in two dimensions. It is also possible to extend the transform into three or higher dimensions.

3.2 The Hough and Radon Transform Connection

In this section it will be shown that the Hough transform is a special case of the Radon transform. The relationship will be proved mathematically to provide a basis for inverting the Hough transform.

The algorithm for the ρ,θ Hough transform is:

```
for each image point (x,y) where there is an edge
  for all θ
    calculate  ρ = x·cos θ + y·sin θ
    increment  A(ρ,θ)
```

This may be written as a mathematical expression as follows:

$$A(\rho,\theta) = \text{no. of x,y points lying on an edge such that } \rho = x\cdot\cos\theta + y\cdot\sin\theta$$

$$A(\rho,\theta) = \sum_x \sum_y (h(x,y) \cdot \delta(\rho - x\cdot\cos\theta - y\cdot\sin\theta))$$

When the function becomes continuous and x, y exist over a plane D:

$$= \iint_D h(x,y)\cdot\delta(\rho - x\cdot\cos\theta - y\cdot\sin\theta)\, dx\, dy$$

$$\text{where } h(x,y) = \begin{cases} 1 & \text{on an edge} \\ 0 & \text{elsewhere} \end{cases}$$

Using the expression derived in the previous section for a two dimensional Radon transform:

$$R(\rho,\theta) = \iint_D f(x,y) \cdot \delta(\rho - x \cdot \cos\theta - y \cdot \sin\theta)\, dx\, dy$$

where $f(x,y)$ is a function describing the intensity at a point (x,y).

If $f(x,y)$ is limited to a binary function, i.e. 1 or 0, this special case of the Radon transform is identical to the Hough transform.

Similarly the Radon transform can be computed using the Hough algorithm by simply substituting the increment step by the addition of the gray level value:

```
for all (x,y)
  for all θ
    calculate  ρ = x·cos θ + y·sin θ
    A(ρ,θ) = A(ρ,θ) + f(x,y)
```

Now the calculation is performed for every x, y point in the image (as opposed to only at edge points for the Hough transform). This leads to greater computation requirements. However, the Radon transform may be computed using the Fourier transform using the method developed in the next section.

It is also interesting to note that when the Radon transform is used to find straight lines in an image it is usual to use an edge detector on the image first. Each pixel value is replaced by the magnitude of the gradient at that point. Therefore, the Radon transform can be calculated in the same manner as the Hough transform, except that, instead of incrementing the accumulator array, the magnitude of the gradient is added. Several

independent authors have made this modification to the Hough transform (Ballard, 1981; O'Gorman and Clowes, 1976) for the purpose of improving the height of the peaks relative to the background, apparently with no theoretical basis. In fact, they were computing the Radon transform. Ballard noted that the strategy of using the gradient magnitude might be more successful for detecting shapes if part of the shape was occluded. Therefore, it is expected that the Radon transform will give better results than the Hough transform.

3.3 Projection Slice Theorem

The Radon (and Hough) transform may be considered as a set of projections through the image at a discrete number of angles.

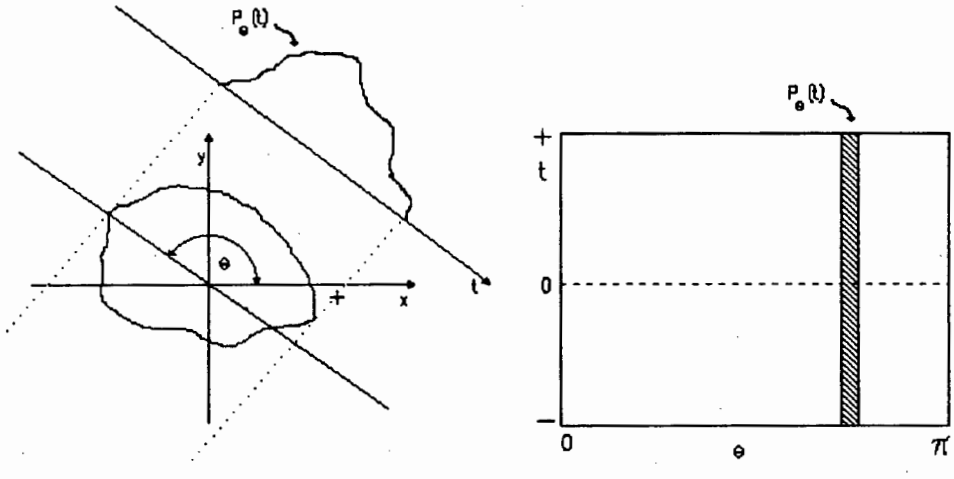


Figure 3.1. A Projection of an image at an angle θ and the corresponding Radon or Hough accumulator.

A projection is a mapping of an N-dimensional function to an (N-1) dimensional function by integrating the function in a particular direction (Mersereau and Oppenheim, 1974). In the case of a Hough transform on a single image there are two dimensions. Therefore the Hough transform is a collection of one dimensional projections. The projections and Hough transform space are shown in figure 3.1.

The projection slice theorem reconstitutes the image from such a set of projections. The theorem states:

The (N-1)-dimensional Fourier transform of a projection is a "slice" through the N-dimensional Fourier transform of f(x) (Mersereau and Oppenheim, 1974).

If $F(u,v)$ is the two dimensional Fourier transform of $f(x,y)$:

$$F(u,v) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x,y)\cdot e^{-j2\pi(ux+vy)}\,dx\,dy$$

and $S_\theta(\omega)$ is the Fourier transform of the projection $P_\theta(\rho)$:

$$S_\theta(\omega) = \int_{-\infty}^{+\infty} P_\theta(\rho)\cdot e^{-j2\pi\omega\rho}d\rho$$

Then the Fourier slice theorem states that:

$$F(\omega,\theta) = S_\theta(\omega)$$

where $F(\omega,\theta)$ represents the values of $F(u,v)$ along a line at an angle θ.

The proof of the above statement (from Rosenfeld and Kak, 1982) is as follows:

Let t, s be the x, y coordinate system rotated by θ. i.e.

$$\begin{bmatrix} t \\ s \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Note that t is equivalent to ρ. Then, from the definition of a projection,

$$P_\theta(t) = \int_{-\infty}^{+\infty} f(t,s)\cdot ds$$

Substituting this into the expression for the Fourier transform of one slice:

$$S_\theta(\omega) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(t,s)\,ds \cdot e^{-j2\pi\omega t}\,dt$$

Transforming this back into x, y coordinates gives:

$$S_\theta(\omega) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x,y) \cdot e^{-j2\pi\omega(x\cdot\cos\theta + y\cdot\sin\theta)}\,dx\,dy$$

$$= F(\omega,\theta)$$

With this relationship proved, it is now possible to recover the image $f(x,y)$ using the inverse Fourier transform:

$$f(x,y) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} F(u,v) \cdot e^{j2\pi(ux+vy)}\,du\,dv$$

The essence of the theorem (in the two-dimensional case) is that the 1D Fourier transform of a projection through the image at an angle θ contains the same information as a slice through the 2D Fourier transform of the image at the same angle, θ (see figure 3.2). This implies that the Fourier transform can be used to find the image, given a set of projections.

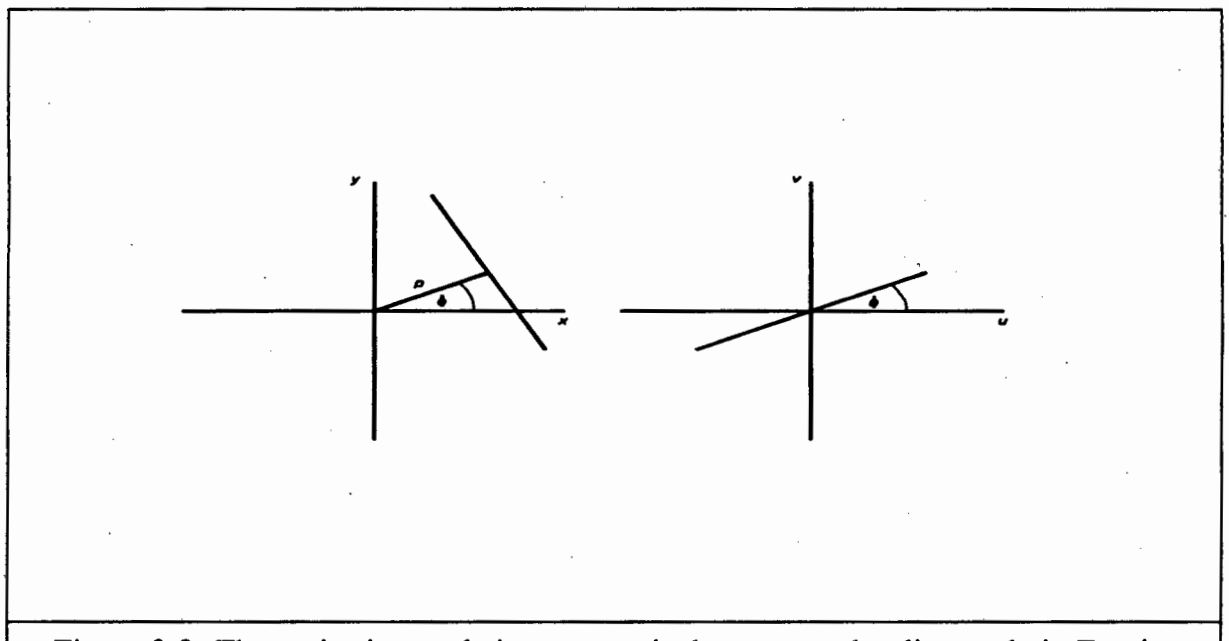


Figure 3.2. The projection angle in xy space is the same as the slice angle in Fourier space. This result follows from the projection slice theorem and is the basis for the reconstruction method.

A further implication of this theorem is that the Radon transform, and hence the Hough transform, can be calculated via the Fourier transform.

3.3.1 Reconstruction from Projections

The above discussion gives a means for inverting the Radon transform. However, it does assume that an infinite number of projections at different angles are available. It also assumes that the Fourier transform is a continuous function.

In practice there must be a limit to the number of projections. This is either due to the storage and computation restrictions, or in the case on Computer Aided Tomography, the limit on the x-ray radiation that the patient can be exposed to. In general, it is impossible to reconstruct an image exactly from a finite number of projections. Exceptions to this are circularly symmetrical images which can be reconstructed from a single projection. However, there are images which fall between these two extremes. If there is some *a priori* knowledge of the image, in particular that there are no resolvable objects of diameter less than r, then the image can be reconstructed from a finite number of projections, N. If the image has a diameter d then (Dudgeon and Mersereau, 1984):

$$N = \frac{\pi d}{r}$$

Another practical factor is that the Fourier transform is usually calculated from the Fast Fourier transform, which is discrete in both space and frequency domains. Therefore, although the Fourier transform of each projection is a continuous function, only a finite number of points are available.

These two factors combine to produce the result that the image cannot be reconstructed exactly. It can, however, be approximated if the image is assumed to be band limited.

The algorithm to perform the approximate reconstruction is (Mersereau and Oppenheim, 1974):

1. Compute the Fourier transform of the sampled projections using a 1D FFT to obtain a collection of sampled slices of $F(\omega_1,\omega_2)$ which represents a polar sampling of $F(\omega_1,\omega_2)$.
2. Estimate the Cartesian samples of $F(\omega_1,\omega_2)$ using interpolation.
3. Estimate $f(x,y)$ using a 2D Inverse FFT.

Mersereau and Oppenheim have a fourth step in their algorithm which convolves the output of the IFFT with a sin(x)/x function. This step has the effect of smoothing the image, but has been omitted here due to its long calculation time and small effect.

The one-dimensional Fourier transforms of each projection should be reasonably smooth for the interpolation to work properly. To avoid discontinuities at the ends, the origin of the Fourier transform is chosen to be at the centre of each projection (since the high frequency component will be small compared with the low frequency as the signal is assumed to be band limited).

The interpolation stage is important to the overall result. Two methods for mapping the polar samples to a Cartesian grid are examined below.

The polar samples are necessarily limited in radius. As these samples are the frequency domain representation of the image, the image must be band limited so that these samples represent all of the non-zero frequency samples (Figure 3.3(a) shows the polar samples). Therefore, each projection corresponds to a band limited 1D function.

The first interpolation method is zero order interpolation. In this case a Cartesian grid is overlaid on the polar samples and each point (x,y) is given the value of the nearest (ρ,Θ) point. The second method is linear interpolation. Here, each Cartesian sample is given a value which is the weighted contribution of the four nearest polar samples. Figure 3.3(b) shows the polar and Cartesian samples. The equation for linear interpolation is given below:

$$c = \frac{(1/d_1)P_1 + (1/d_2)P_2 + (1/d_3)P_3 + (1/d_4)P_4}{(1/d_1)+(1/d_2)+(1/d_3)+(1/d_4)}$$

Higher orders of interpolation may be used. Instead of linear interpolation, a polynomial function would be used. There are two approaches to using higher order functions. The first is to obtain a higher degree of accuracy for the interpolated function without ensuring that the function has continuous derivatives. The second is to force continuity of some of the derivatives while sacrificing some accuracy (Bicubic interpolation or bicubic spline) (Press et al, 1986).

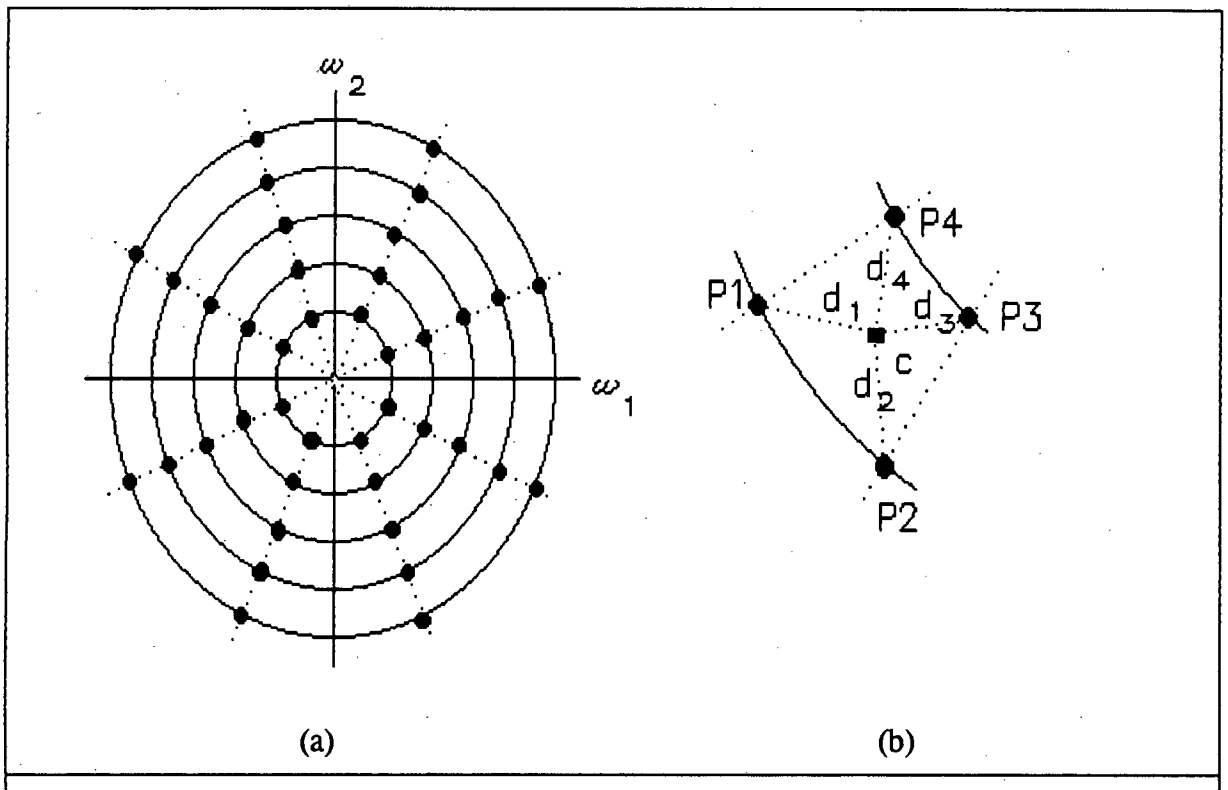


Figure 3.3. Polar sampling of the frequency domain. (a) Polar samples of $F(\omega_1, \omega_2)$. (b) The Cartesian sample is the weighted sum of the polar samples.

3.4 Filtered Back Projection Algorithm

The Filtered Back Projection algorithm is a space domain method (as opposed to the Projection Slice Theorem which operates in the frequency domain). It is the method used in most commercial CAT scan devices because it is extremely accurate and can be implemented for fast execution (Rosenfeld and Kak, 1982).

If the Radon transform is written as (Rosenfeld and Kak, 1982):

$$P_\theta(t) = \iint f(x,y) \cdot \delta(x \cdot \cos\theta + y \cdot \sin\theta - t)\, dx\, dy$$

Then the inverse Radon transform, as derived by Rosenfeld and Kak (1982), is:

$$f(x,y) = \int_0^\pi Q_\theta(x \cdot \cos\theta + y \cdot \sin\theta)\, d\theta$$

where Q_θ is the projection P_θ filtered by a one dimensional filter:

$$Q_\theta(t) = \int_{-\infty}^{\infty} S_\theta(\omega)\,|\omega|\, e^{j2\pi\omega t}\, dw$$

As the data is discrete, this expression must be approximated as (Hinkle et al,1986):

$$f(x,y) \approx \frac{\pi}{M} \sum_{n=1}^{M} Q_{\theta n}(x \cdot \sin\theta_n + y \cdot \cos\theta_n)$$

The algorithm for computing this summation is:

1. For each projection P_θ
   Filter to form Q_θ
2. For all points x,y
   For each filtered projection, Q_θ
   calculate $k = x \cdot \sin\theta + y \cdot \cos\theta$
   image(x,y) = image(x,y) + $Q_\theta(k)$
3. Scale results by a factor π/M

The filtering on the projections is a high pass filter, emphasizing the high spatial frequencies of the projections. This will have the effect of emphasizing any high frequency noise which inevitably results from the digital quantisations. The response of the filter is shown in figure 3.4. As the projections themselves are bandlimited the frequency response of the filter above a certain cutoff frequency, f_c, is insignificant (Dudgeon and Mersereau, 1984). Therefore the filter response can be made zero for frequencies above f_c. Also, resolution can be traded off against noise performance by lowering this cutoff frequency.

This can be done by windowing the response in figure 3.4 with bandlimiting window functions. Several of these are: the Shepp-Logan filter, the lowpass cosine filter, and the generalised Hamming window (Hinkle et al, 1986). The different window functions give different trade-offs between the noise and image resolution. Hinkle et al found that the lowpass cosine filter produced the most visually pleasing results.

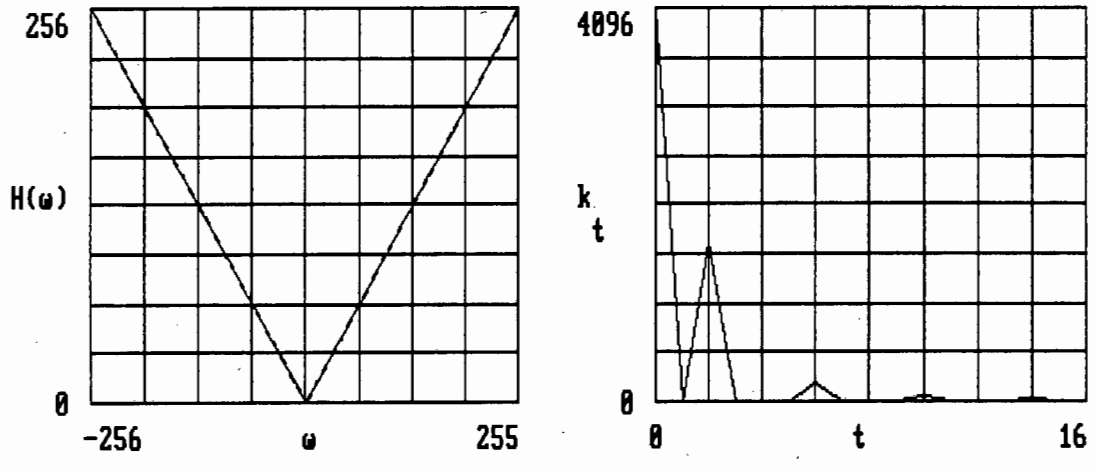


Figure 3.4. Frequency and time responses of filter used to form filtered projections.

Therefore, using the fast Fourier transform:

$$Q_\theta(t) = \mathrm{IFFT}(\ \mathrm{FFT}(P_\theta(t)) \cdot H(\omega) \cdot \text{window fn})$$

where:

$$H(\omega) = |\omega| \cdot b_w$$

Sometimes it is faster, or at least as fast, to implement this in the space domain. This is true when there is dedicated hardware for performing convolutions. In the time domain the above expression is:

$$Q_\theta(t) = \int_{-\infty}^{\infty} P_\theta(t') \cdot h(t-t')\ dt'$$

Since the projections exist only over a limited radius and are only available at discrete samples:

$$Q_\Theta(n\tau) = \tau \sum_{k=0}^{N-1} h(n\tau - k\tau) \cdot P_\Theta(k\tau)$$

Where $h(t)$ and $h(n\tau)$ are the time domain responses of the bandlimited $H(\omega)$ windowed by a smoothing window.

3.5 Artifacts in the Reconstructed Images

Both the filtered back projection algorithm and the projection slice method assume that the projections are bandlimited but, when these algorithms are used to invert the Hough transform, this may not always be strictly true. The Hough transform is used on binary images, which inherently must have steps (i.e. from 1 to 0 or 0 to 1). These steps also occur in images operated on by a Radon transform, but they tend to be smaller (i.e. a gradual change from light to dark over, say, a range of 0 to 255). The steps in a binary image are larger and generate more high frequency components. So, although the image is bandlimited, a large percentage of the signal energy is at high frequencies.

Two artifacts that appear in the reconstructed images are Moiré patterns and Gibbs phenomena. There are also two causes for artifacts, namely, the finite number of projections and aliasing. Since backprojection is a linear process the final image can be thought of as a sum of components. One component is the image made from band limited projections, degraded by the finite number of projections. The second component is the image made from the aliased portion of the spectrum in each projection (Rosenfeld and Kak, 1982).

The streaking artifacts are caused by the aliasing errors in the projection data. These are the Gibbs phenomena and reduce with increasing number of samples per projection.

The fringing patterns arise from the insufficient number of projections. These are the Moiré patterns.

3.6 Comparison of methods

For an image of dimensions NxN, the projection slice theorem requires N 1D FFTs followed by interpolation, followed by a two dimensional NxN IFFT. Since a one dimensional FFT requires $N\log_2 N$ calculations the total number of computations can be calculated. Similarly the requirements for the filtered back projection algorithm can be calculated.

Projection slice method		Filtered Back Projections	
N 1D FFTs:	$N(N \log_2(N))$	Filtering:	N^2
Interpolation:	N^2	Summation:	N^3
NxN 2D IFFT:	$N(2\ N \log_2(N))$	Scaling:	N^2
	======		=====
Total:	$3N^2\log_2(N) + N^2$	Total:	$2N^2 + N^3$

Table 3.1 Comparison of calculations required for the projection slice method and filtered back projection algorithm.

Therefore, for $N>8$ the projection slice theorem is more efficient. For $N=512$ it is a factor of approximately 18 times more efficient. However, the filtered back projection algorithm does have some advantages.

Firstly, apart from the discrete nature of the data, no approximations are necessary. Therefore it should be more accurate than the projection slice method which requires approximation in the interpolation stage. Secondly the Fourier transform generates real and imaginary components of data which means that twice as much intermediate storage is needed for the projection slice method. Thirdly the filtered back projection method operates in a strictly sequential manner. The fourth advantage is that the same, simple, calculation is carried out at each image point. These last three factors make it very suitable for DSP chip implementations as well as for parallel processing.

CHAPTER 4

4. Implementation

The previous two chapters presented the theoretical background for the implementations discussed here. This chapter deals with the practical issues of implementing the theory.

The theoretical model for the analysis of an image with the Hough transform is: A forward transform, filtering, and finally an inverse transform. If the original image is binary, the forward transform is the Hough transform. In the case of a gray level original image the Radon transform is used. The filtering that is done depends on the application. The inverse transform is the inverse Radon transform (or reconstruction from projections).

The implementation of the Hough, Radon and inverse Radon transforms on three architectures will be described. The architectures described were not chosen particularly because of their suitability to these problems, but rather because they were available. Although the results cannot be expected to be as impressive as those from dedicated hardware, the equipment used is available and is relatively inexpensive.

The three architectures are: MicroVAX II, IBM PC/AT 386, and the Texas instrument TMS320C25 DSP processor. The three environments vary greatly according to the amount of memory accessible and parallel processing capabilities.

The formats of the files were the same in all cases. The input image was 512 by 512 pixels. For gray level images, 8 bits per pixel were used. For binary images only 1 bit per pixel is needed. The output files were in the same format. In the cases of the Hough and Radon transforms 512 projections over a diameter of 512 pixels were taken. Each projection was stored as a column in the output image.

The algorithms that will be discussed for each architecture are the Hough, Radon, Two Dimensional Fast Fourier Transform (2D FFT) and its inverse, the Two Dimensional Inverse Fast Fourier Transform (2D IFFT). Two inverse Radon transforms, the Filtered Back Projection Algorithm, and the Projection Slice Theorem will also be discussed.

The 2D FFT and 2D IFFT are discussed because they are needed for the Projection Slice Theorem. They can also be used for calculating the Hough or Radon transform.

4.1 MicroVAX

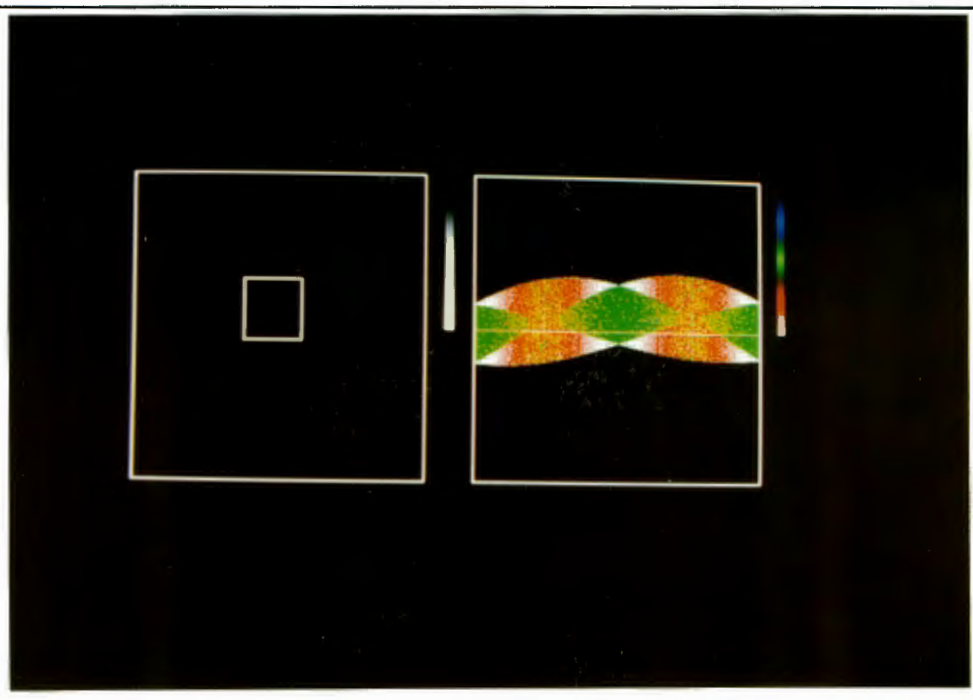
The MicroVAX II is a multi-user minicomputer. It has a single main processor with an additional floating point unit. There are nine megabytes of physical memory installed, but the operating system allows access to up to 4 gigabytes of virtual memory. This is provided by mapping pages to and from the disk, but is totally transparent to the user.

With these parameters it can be seen that memory is not a limitation, although one should take care not to exceed the 9 M Bytes to avoid disk accesses. The floating point unit means that there is little to be gained by converting floating point operations into integer arithmetic.

4.1.1 Hough and Radon Transforms

The Hough transform can be implemented as the algorithm given earlier with only one modification. That is, to precompute the values of sin θ and cos θ that will be needed, and to store these in a lookup table. The algorithm is then:

```
1. Load binary image
2. Clear Hough array, A
3. For all θ
     costable[θ] = cos θ
     sintable[θ] = sin θ
4. For all y points
     For all x points
          If image[y,x]<>0 then
               For all angles θ
                    ρ=round(x·costable[θ]+y·sintable[θ])
                    A[ρ,θ] = A[ρ,θ] + 1
5. Store Hough array
```



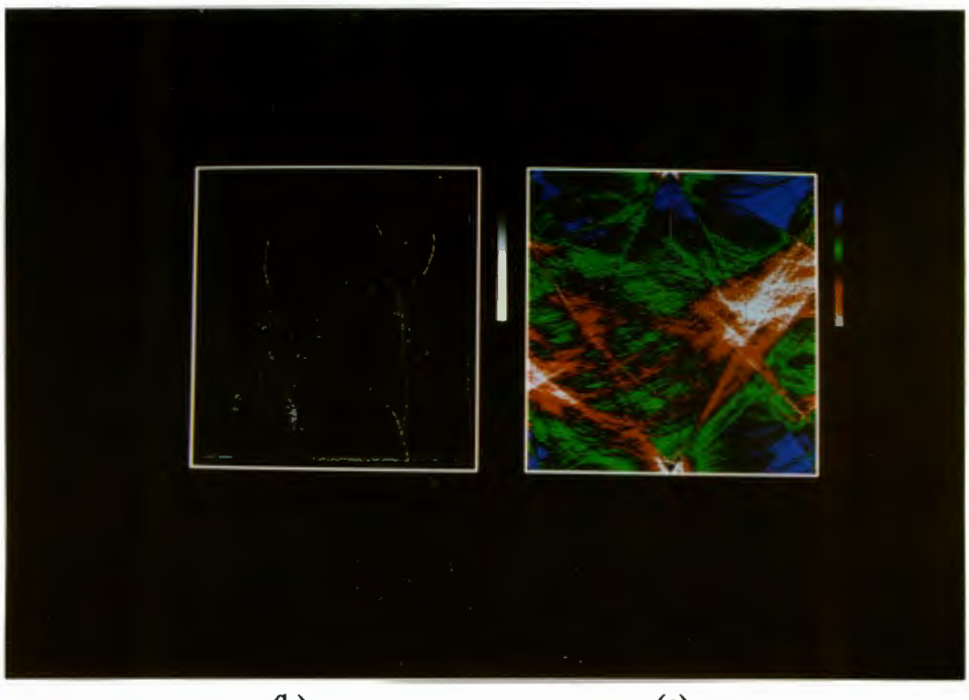
(a) (b)

Figure 4.1. Hough transform of a square. (a) Square. The image is binary, with 1 on the edges and 0 elsewhere. (b) Hough transform of square. The image is displayed in false colour to improve the contrast.

In the programs used here, the input images were all 512 by 512 pixels by 256 levels. The output images were the same format, so the Hough space consists of ρ in the range [-256, 255], and 512 discrete values of θ in the range $(0,\pi)$. Each bin in the accumulator array must be capable of holding all the votes contributing to it. Therefore, for a 512 by 512 image the longest line is $(\sqrt{2} \cdot 512) = 724$ pixels. As there may be some contribution from background noise the maximum should be larger than this to avoid overflow. In practice, the VAX can store either a byte value (0..255) or a 32-bit integer. If simple images, with short line segments, are used byte values are sufficient. This saves both memory and time (because the results do not need to be scaled back to 8 bits for storage). Figure 4.1 shows the Hough transform of a square.

The Radon transform is very similar to the Hough transform. The algorithm is given below:

(a)



(b) (c)

Figure 4.2. The Radon transform of an image. (a) Original image - GIRL. (b) This image is the magnitude of the edges in the original GIRL image. (c) The Radon transform of (b). Again false colour is used.

```
1. Load gray level image
2. Clear Radon array, R
3. For all θ
     costable[θ] = cos θ
     sintable[θ] = sin θ
4. For all y points
     For all x points
          If image[y,x]<>0 then
               For all angles θ
                    ρ=round(x·costable[θ]+y·sintable[θ])
                    R[ρ,θ] = R[ρ,θ] + image[y,x]
5. Scale results to 8 bits
6. Store Radon array
```

Now, however, it is important to use sufficient bits for the Radon array as each vote does not simply increment the array, but adds a gray level to it. Note that there is now a scaling of the results back down to eight bits. In general the input to the Radon could be any gray level image. Since the Radon transform is being used to find lines, the input to the Radon transform is an edge-magnitude image. An example of this type of image would be the result of convolving an image with the Sobel operators and then ignoring the directional information.

4.1.2 2D Fast Fourier Transform

The IEEE Fortran math routines contain optimised one dimensional fast Fourier transforms (1D FFT). Using these, the algorithm for a two dimensional Fourier transform is:

```
1. Multiply each element (x,y) by -1^(x+y)
2. Take 1D FFT of each row
3. Transpose resulting matrix
4. Take 1D FFT of each row again
```

The first step is necessary to move the origin from the corner of the image to the centre. Without this step, the 2D FFT has its zero frequency in each corner. The 2D FFT will be

used in the projection slice method to reconstruct an image from its projections. In this method the zero frequency point must be in the centre of the image.

The transposition can be done by moving data from one array to a new array. On the VAX there is enough memory for this. The 2D inverse FFT would be the above, in reverse order, using 1D inverse FFTs instead of the 1D FFTs.

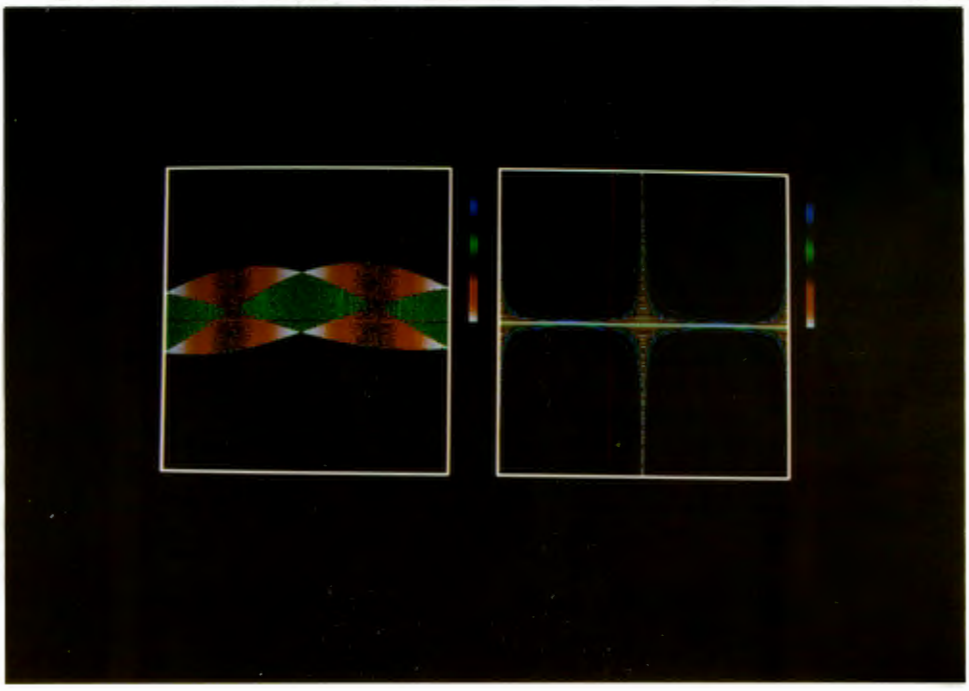
4.1.3 Reconstruction from Projections

The inverse Radon transform (reconstruction from projections) is discussed in this section. The inverse Radon transform can be implemented using the projection slice theorem (PST) or the filtered back projection algorithm. It has been shown that the projection slice method is more efficient. As the projection slice method uses the Fourier transform in one and two dimensions, a large memory space is needed. Additionally, floating point calculations are required. The VAX architecture is suitable from both these viewpoints.

The FFT routines take as input, and output, floating point variables which are 5 bytes each in VAX Pascal. Therefore the array needed to store the results after stage 2 must be 512 by 512 by 5 by 2 (for complex data) bytes, which is 2.5 M Bytes. A second array of the same size is needed to hold the interpolated results. Therefore 5MB of memory is needed at any one time. However, as floating point arithmetic is used there is no perceivable truncation noise.

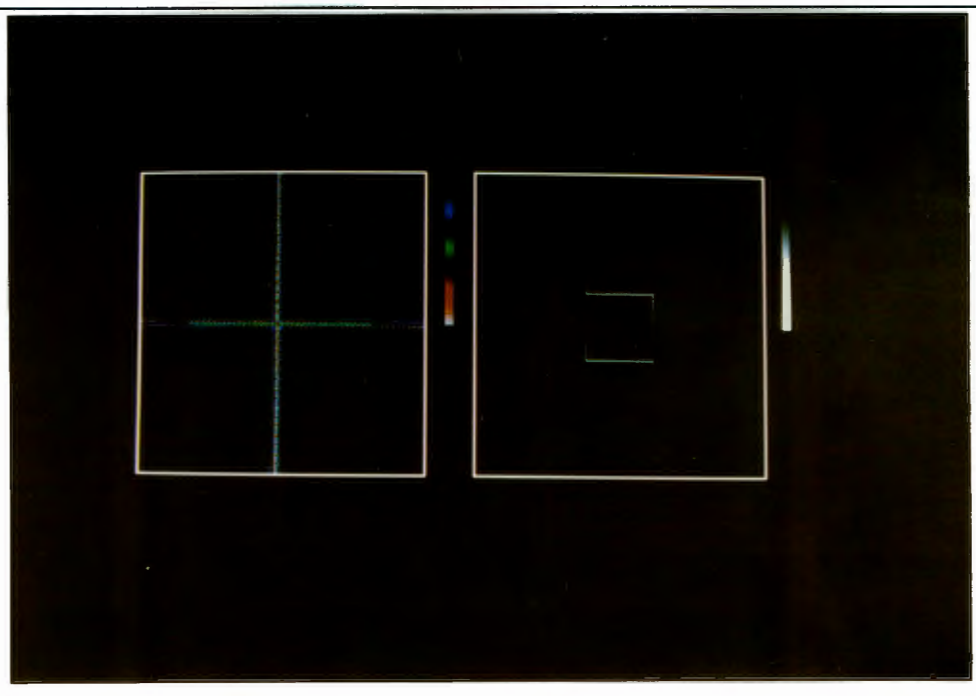
The algorithm for the inverse Radon transform (PST) is:

1. Load file
2. Take 1D FFT at each angle, θ
3. Interpolate onto Cartesian grid
4. Take 2D IFFT
5. Scale results
6. Store file



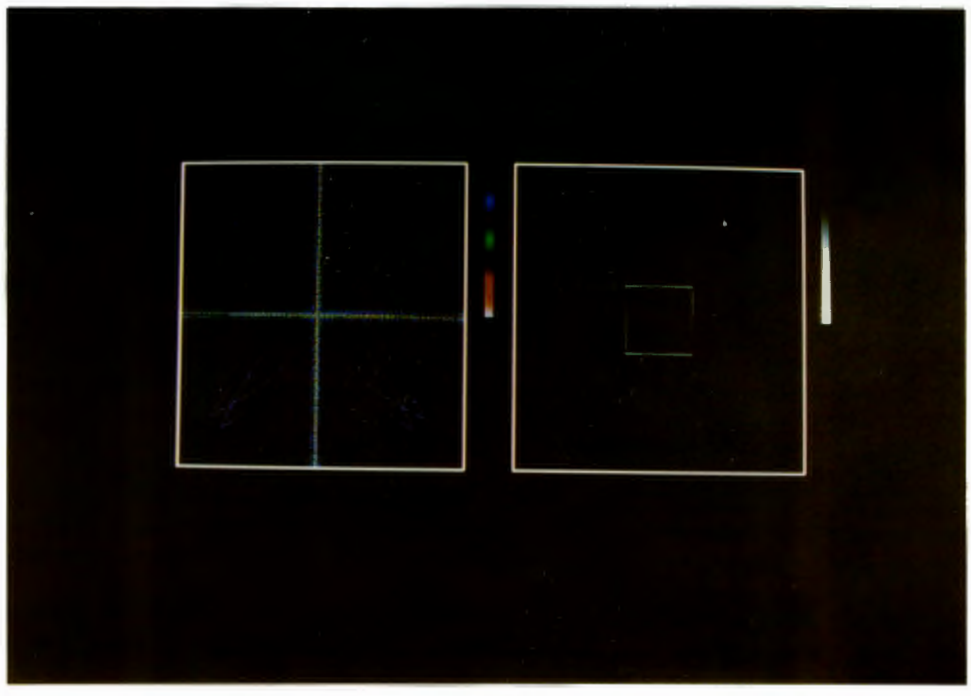
(a) (b)

Figure 4.3. Initial stages in the reconstruction from projections. (a) Hough transform of the square in figure 4.1. (b) One dimensional column FFTs of (a).



(a) (b)

Figure 4.4. Final stages for zero order interpolation. (a) Zero order interpolation of 4.3(b) onto Cartesian grid. (b) Two dimensional IFFT of (a).



(a) (b)

Figure 4.5. Final stages for linear interpolation. (a) Linear (first order) interpolation of 4.3(b) onto a Cartesian grid. (b) Two dimensional IFFT of (a).

In the above images, figure 4.3 shows the initial stages, figure 4.4 shows the final stages for zero order interpolation and figure 4.5 shows the final stages for linear interpolation. Note that figures 4.4(a) and 4.5(a) represent the frequency domain. The origin is in the centre of the image, and high frequency at the edges. Compare the spreading at high frequency in figure 4.4(a) to that in figure 4.5(a). This spreading causes lower image resolution when using the zero order technique in the final reconstruction.

The forward Radon transform can be implemented by simply reversing the order of each operation in the above algorithm.

4.2 IBM PC/AT/386

In principle any IBM compatible PC/AT or 386 computer can be used. It must, however, have extra memory conforming to the Expanded Memory Standard 4.0 (EMS 4.0). The original PCs' (Intel 8088 based) had a maximum address space of 1Mb, and the newer

computers (80286 and 80386 based) are used in a mode which emulates the 8086. The EMS scheme was introduced to overcome this limitation by allowing access to 16Mb of memory. This scheme allows memory. This scheme allows the mapping of 16Kb 1Mb address space. However, the operating system does not automatically take care of this, and the programmer must keep track of what lies in each page. Therefore, although the PC can use the extra memory, it is not as easy to use as is the case on the VAX.

The PC used for the results here has 2Mb of memory, no floating point processor and a 80386 processor. This represents the cheapest of the three implementations discussed in this chapter, but it does have several limitations. The first has already been mentioned, the memory management. The memory is divided into 1Mb for program and data, and 1Mb of expanded memory for data accessible in 64 blocks of 16K bytes. It takes a finite amount of time to swap pages, but the 80386 has memory management capabilities built into the processor so the page swapping is fast. On the other processors the mapping must be done either by specialised hardware or by software (in which case it will be slow). Therefore the 386 is suitable for this application. The second is the lack of a floating point processor, which means that wherever possible calculations should be limited to integer operations.

4.2.1 Hough and Radon Transforms

The implementation of the Hough transform is the same as on the VAX. As before, one byte is used for each cell of the accumulator. For the Radon transform a long integer (32 bits) is needed for each cell, and so 1Mb is needed altogether. Therefore the expanded memory must be used to hold the accumulator array. As the access to the accumulator is not sequential many page requests will be needed, slowing down the execution significantly unless a 80386 processor is used.

Additionally the calculation of:

$$\rho = x \cdot \cos\theta + y \cdot \sin\theta$$

is turned into an integer calculation. The values of cos and sin are precomputed and stored in a lookup table. In order to use integer values they must be multiplied by some scalar value. That is:

```
costable θ = (cos θ) · 128
sintable θ = (sin θ) · 128
```

and:

```
ρ = (x·costable θ + y·sintable θ) / 128
```

Using a scale factor of 128 means that there are only 128 discrete values for costable θ or sintable θ. This means that accuracy is lost. It would be desirable to increase the scale factor, which would in turn decrease the quantisation error. However, a value of higher than 128 will cause the intermediate result (before the division) to overflow 16 bits. This causes an error unless the compiler is forced to use more bits for the intermediate result. If it is forced, the calculation time is much longer.

4.2.2 2D Fast Fourier Transform

The implementation of the 2D FFT is significantly different. Floating point arithmetic is inconvenient to use for two reasons: memory requirement and slow calculation. Therefore a one dimensional integer FFT was written for the PC. An integer is represented by 16 bits. If the input to the FFT is 8 bits, the output will fit within 16 bits. This makes the calculation times shorter, and only requires 512 by 512 by 2 (complex data) by 2 (16 bits per sample) bytes (1Mb) for the FFT array in the 2D case. However, the input must always be scaled to the correct range for the FFT routine. This adds to the number of computations.

The 2D FFT requires the data to be transposed. This must either be done in-place, or add an extra megabyte of memory to the system. It is possible to transpose a matrix in-place in the most efficient way by using Eklundh's transposition algorithm (Lim, 1990).

This algorithm is as follows. First, divide an NxN matrix into four N/2 x N/2 blocks. Then swap the first row of the upper right block with the first row of the lower left block. Repeat this for all of the rows in the block. Then, repeat the process, dividing the matrix into 16 N/4 by N/4 blocks and so on until the block size is one cell. At this stage the matrix will be transposed. The number of stages required is $\log_2 N$ and $N^2/2$ swapping operations are required for each stage. This algorithm is used for 2D FFTs when the results must be stored on disk because it is optimised for the minimum number of disk accesses. The situation with expanded memory is rather similar to storing the data on disk

in that the data in expanded memory can only be accessed in blocks. Additionally it is desirable to minimise the number of read and write operations on the data in expanded memory because of the finite time taken to swap pages. Therefore, using Eklundh's algorithm on data stored in expanded memory minimises the number of accesses to this memory.

In practice the efficiency of each stage is different, becoming worse as the block size decreases. This appears to be due to the fact that it is more efficient to move large quantities of data at once, rather than in small blocks. The time taken to transpose the matrix is a significant portion of the time taken for the entire 2D FFT.

The complete algorithm for a 2D FFT is:

1. 1D FFT of each row
2. Scale results
3. Transpose
4. 1D FFT of each row
5. Shift to centre
6. Scale results

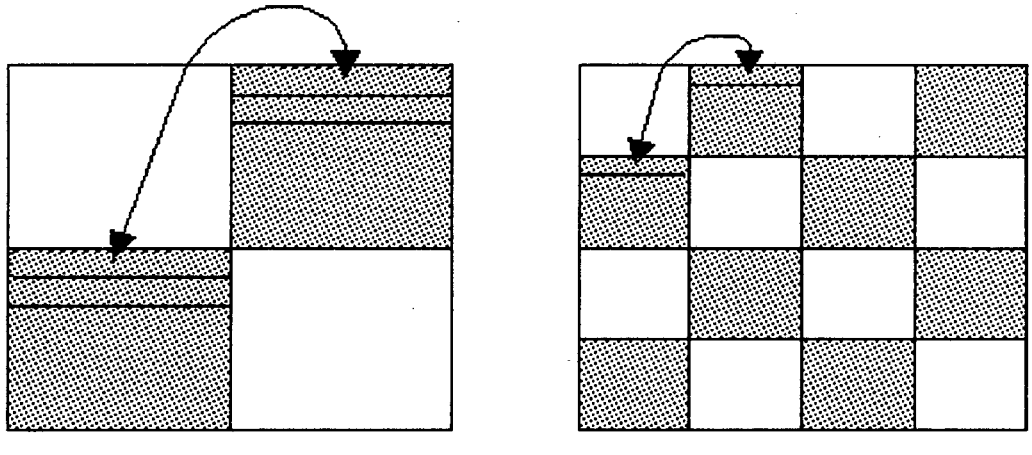


Figure 4.6. Eklundh's transposition algorithm. The arrows in the diagram on the left show the rows that are swapped. On the right is the next stage of the algorithm, where the block size is halved.

The fifth step is necessary to move the zero frequency point from the corners of the image to the centre. In the MicroVAX implementation this was done by multiplying each element by $-1^{(x+y)}$ before the 2D FFT. Here, it is faster to move the data than to multiply by this factor.

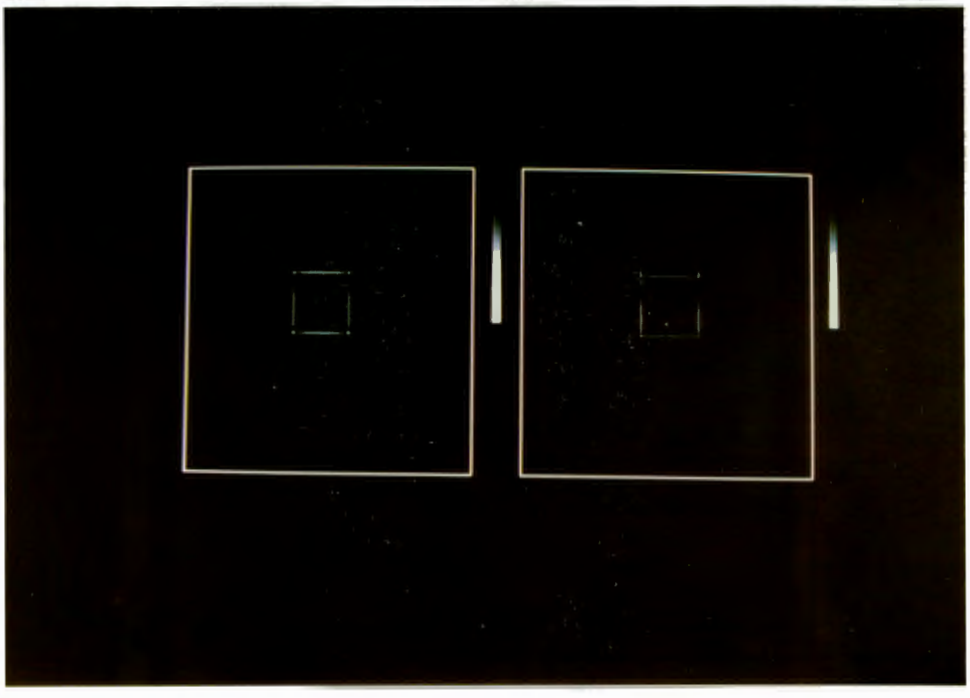
4.2.3 Reconstruction from Projections

The Fourier based methods for reconstruction are less suitable than space domain methods because integer arithmetic is preferable on the PC. They can be used, but many scaling steps are needed for the FFT routines. Therefore the filtered back projection algorithm is more suited to this architecture.

The filtered back projection algorithm needs an array the same size as the original image (512 by 512 pixels) but the dynamic range required is larger. On the PC a 32-bit signed integer is used to store each pixel value. The results are scaled back to 8 bits for storage and display. The one dimensional filtering of the projections is implemented in the space domain. The filter is as described in chapter 3, windowed by a lowpass cosine. Fortunately, some of the filter coefficients are zero, so implementing this as a convolution does not require many multiplications. For example, a 13-tap filter only requires 7 multiplications.

The projection slice theorem can be implemented on this architecture, but as mentioned above, many scaling steps are required. These significantly affect the overall calculation time. Additionally, the use of integer arithmetic causes truncation errors that degrade the results.

Figure 4.7 shows the square reconstructed using the filtered back projection algorithm and the reconstructed image using the projection slice theorem. The original image was the same square as used in section 4.1.



(a) (b)

Figure 4.7. Reconstructed images using the FBPA and PST on an IBM/PC/386. (a) Reconstruction using filtered back projection algorithm. (b) Reconstruction using projection slice theorem.

4.3 Digital Signal Processor

The Texas Instrument TMS320C25 digital signal processor was used. The processor, together with 8 Kbytes of memory, is mounted on a plug in card for the IBM PC. The card is manufactured by Dalanco Spry. The card also features D/A and A/D converters for audio frequencies, but these are not used here.

As the card plugs into a PC, the memory management problems still exist. However, now it is possible to use the host processor solely for memory management and to dedicate the DSP to applying the algorithms. It is also possible to plug more than one DSP card into the host computer and execute algorithms in a parallel environment. The limitations of this architecture are that there is still no floating point processor, and that the TMS processor only has direct access to 8 kbytes of memory.

The projection slice theorem can be implemented on this architecture. Integer FFTs are again used. This leads to problems with dynamic range, but with careful scaling the results are acceptable.

4.3.1 Hough and Radon Transforms

Both of these transforms require access to non-sequential elements in the transform space. Therefore the DSP processor is not particularly suited to this application.

Additionally, no compiler was available for the TMS DSP processor. For this reason it was not possible to investigate further in the time available.

4.3.2 2D Fast Fourier Transform

The two dimensional FFT can be implemented reasonably well on this architecture. The TMS320C25 is not suited to floating point arithmetic, so integer FFTs are used. Such a FFT program was available. This still implies that the output must be scaled, but the host processor can be used for this. The DSP processor is used for the actual FFTs while the host processor does the scaling and transposition.

The steps in the 2D integer FFT are as before except that the FFTs are performed by an assembler routine supplied with the Dalanco Spry card. The routine performs a 512 point complex FFT in approximately 20 ms. The rest of the algorithm is performed by the host processor.

Eklundh's algorithm is used again to transpose the matrix after the first FFTs. Here, the DSP processor cannot be used because it cannot access large memory spaces.

Figure 4.8(a) shows the algorithm as for the PC alone. Figure 4.8(b) shows the modified version for the DSP and PC together. Once the first row of the input image has been scaled the DSP processor can start a FFT. In parallel the host processor scales the next row. This continues until all rows have been processed. The matrix is then transposed. No parallel calculations can be performed here. Once transposed the above FFT process is repeated.

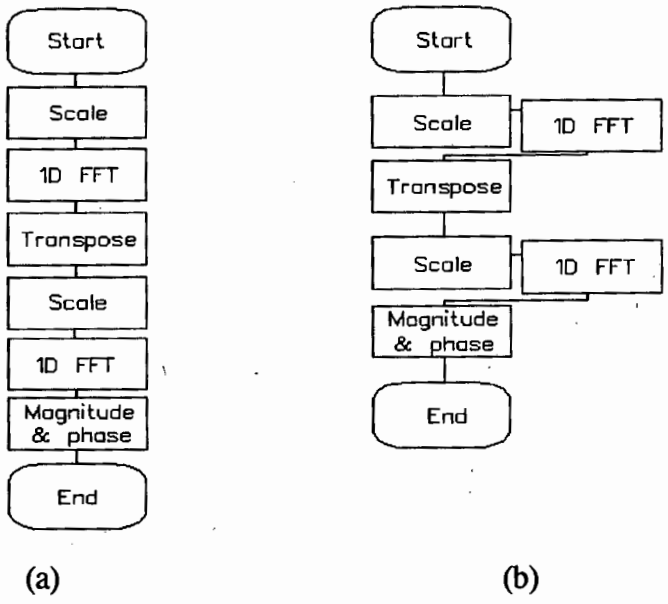


Figure 4.8. Flowchart for a Two dimensional FFT. (a) Sequential FFT. (b) Parallel processes

In principle, more than one DSP card could be used to perform FFTs. Careful attention would have to be given to scaling performed by the host processor, as the slowest process will limit the overall processing time.

4.3.3 Reconstruction from Projections

The algorithm is the same as for the MicroVAX. The TMS processor is only used for the FFT computations with the rest of the algorithm being implemented on the host processor. Therefore, the slowest stage is the interpolation. This limits the overall speed of the program.

4.4 Timing comparisons

The approximate timing for each of the algorithms running on the different architectures is given in table 4.1. The following times are in minutes and are approximate. They are only intended to give an idea of the relative performance of each architecture.

The MicroVAX architecture gives reasonable performance and it is easy to write programs because of the large memory space and floating point processor. However, it is expensive. The PC architecture is limited by its memory management. However it is a relatively cheap solution. The DSP processor is also limited by memory. The DSP processor can only access the small amount of memory on the plug-in card. The host PC must manage the rest of the memory. However, this solution provides the capability of parallel processing.

Architecture	Time 2D FFT	Time Hough	Time Radon	Time FBPA	Time PST
Micro VAX	5	1	1	150	7
PC/AT 386	3	0.1	0.9	17	9
Texas DSP	0.9	-	-	-	6

Table 4.1. Comparison of time taken (in minutes) for different architectures. The image used for testing was the SQUARE. 2D FFT = Two Dimensional Fast Fourier Transform, FBPA = Filtered Back Projection Algorithm, PST = Projection Slice Theorem.

A PC host computer, with several DSP processors each with its own large memory (approximately 512 kB) would provide an economic, yet effective solution. In this case the host computer would split the problem into several tasks and program each DSP processor to perform one task. At the end it would collect the results from each DSP.

An architecture proposed by Hinkle et al (Hinkle et al, 1986) is the Parallel Pipeline Architecture. The authors called this architecture the Parallel Pipeline Projection Engine (P^3E). It was specifically designed for computing the Radon transform and its inverse (filtered back projection). They suggest pipelined processors, rather than parallel ones. Their paper does not mention typical computing times.

Another approach to computing the Hough transform was suggested by Ben-Tzvi and Sandler (1989). Their approach is interesting because they use analog multipliers and a summer to calculate the term:

$$\rho = x \cdot \cos\theta + y \cdot \sin\theta$$

The result is converted to a digital number by a flash A/D converter. The authors expect computation times for the Hough transform in the order of milliseconds. In addition it is possible to modify the circuit to perform the filtered back projection algorithm. A hybrid VLSI device could replace the discrete integrated circuits.

Several other authors have designed VLSI implementations for the Hough transform. Rhodes et al (1988) designed a monolithic Hough transform processor using restructurable VLSI which is a wafer-scale integration technology. The wafer is manufactured with uncommitted interconnections which can be made later with a laser. The authors manufactured several devices of this type and found that the processing time for an image with 10% edge points was approximately 20ms.

Another real time processor was designed by Hanahara et al (1988). This processor used discrete logic and memory devices to achieve transform times in the order of one second. The devices used were standard TTL logic and CMOS static RAMs. Therefore, this type of processor could be built without special manufacturing equipment (such as lasers).

These hardware implementations of the Hough transform show that it is possible to reduce the computation time significantly. Since the calculations required for inverting the Hough transform via the filtered back projection algorithm are virtually identical to those required for the forward transform, these hardware designs can be adapted to perform the inverse transform. Two such processors, one for the forward transform and one for the inverse transform, would make it possible to implement the transforms in real time.

CHAPTER 5

5 Applications

This chapter presents methods for solving image processing problems using the theory and practical implementations described in the previous chapters. Results are presented for both ideal and noisy conditions.

The basic outline for the methods in this chapter is: A forward transform, filtering, and an inverse transform. These stages correspond to: the Hough or Radon transform; filtering which is typically a convolution with a 3x3, or larger, mask; inverse Radon transform using either the filtered back projection or projection slice methods.

The results are given for two areas of image processing, namely, image enhancement and image analysis. Within each of these areas examples of the uses of the Hough or Radon transforms and their inverse are given. These methods are applied to the problem of extracting information about scenes containing polyhedra.

The above methods all apply a filter to the Hough parameter space. The concept behind filtering in Hough space is similar to that behind filtering in the frequency domain. To filter in the frequency domain the Fourier transform is used to convert the spatial representation of the image to a frequency representation. In this domain, it is easy to design filters which select specific frequencies of interest. The filter can be applied to the transformed image before inverting the Fourier transform to return to a spatial representation.

In the same way the Hough transform is applied to an image to transform it into Hough space. The Hough transform converts a difficult global detection problem in image space into a more easily solved local peak detection problem in parameter space (Illingworth and Kittler, 1988). In this space the coordinates are the parameters of features in this image. In the case of the ρ, Θ transform the coordinates are ρ and Θ and any straight line in the image generates a peak in the transform domain. These peaks are easier to detect than the original lines and it is also easier to design filters that will select features of interest. Such features will become visible once the transform is inverted.

A few assumptions will be made for this section. Firstly, the version of the Hough transform proposed by Duda and Hart for detecting straight lines will be used. Secondly, since the Hough transform operates on a binary, edge magnitude picture it will be assumed that the input gray scale image has already been subjected to an edge detection operator, and has then been thresholded to obtain a binary image. Additionally, since the Hough transform has been shown to be a special case of the Radon transform, the Radon and Hough transforms can be used interchangeably, except that the Hough transform will only work on a binary image.

When the ρ, θ Hough transform has been applied to an image the accumulator array forms a space with features corresponding to the lines in the image. The feature generated from a straight line segment in the image is a butterfly shape. Figure 5.1(a) shows a typical butterfly. Figure 5.1(b) shows the same peak, but on a three dimensional plot.

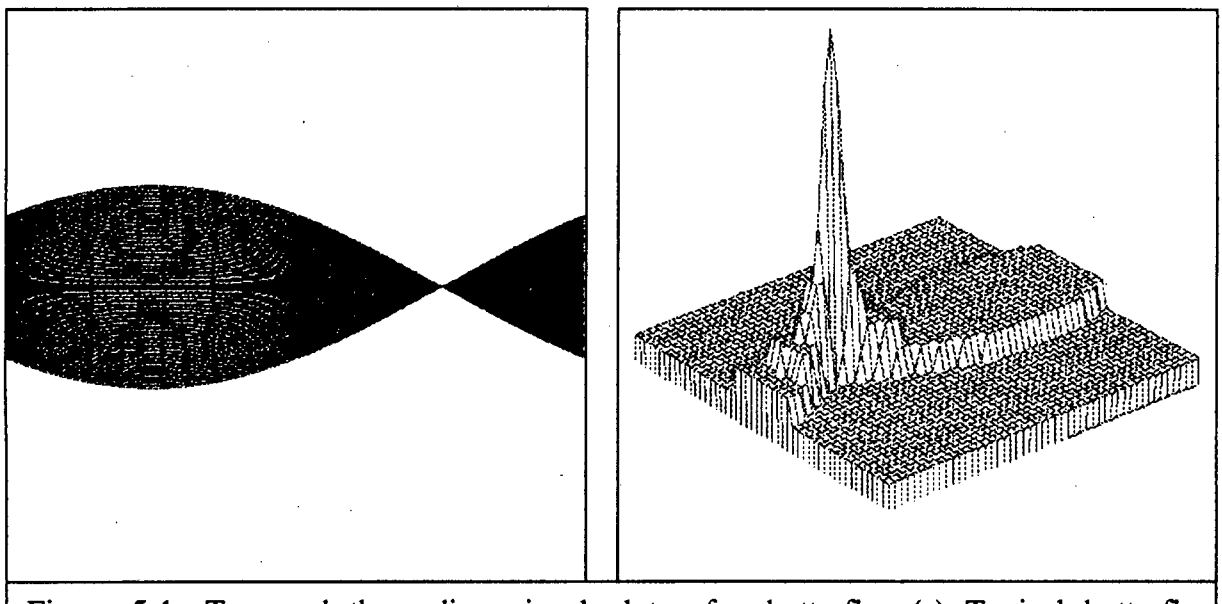


Figure 5.1. Two and three dimensional plots of a butterfly. (a) Typical butterfly resulting from the Hough transform of a straight line. (b) 3D plot of butterfly.

The usual technique is to find the position of the peak at the centre of the butterfly. The ρ, θ coordinates of this point specify a line in the image. Normally this is the only information extracted from the image. The techniques discussed here take the entire shape of the butterfly feature into account in order to extract more information.

The final section investigates the performance of the algorithms in the presence of additive white gaussian noise.

5.1 Image Enhancement

The Hough, or Radon, transform and the inverse can be used to enhance features in an image. In the case of image enhancement the aim is make features in an image more visible to a human observer. Therefore, since the input image is a gray level image, it is usually more applicable to use the Radon transform.

In either case (Hough or Radon), the straight line transform is used so the features that can be enhanced are straight lines. The generic operation would be: forward transform, filter, inverse transform. In this section, the types of filtering applicable to image enhancement will be discussed.

Such an example occurs in the image generated by synthetic aperture radar (SAR) shown in figure 5.3. This image was generated by the space shuttle passing over the Grahamstown area. It was noticed that there were lines in the image which were probably electrical power lines. In the Radon transform of this image peaks indicate this position of the lines. To enhance the lines, these peaks must be enhanced. The filtering could be any operator which increases the height of the peaks and decreases the depth of the troughs in the transform space (Murphy, 1986).

An edge detector was used to extract the edges from the image and then transformed using the Radon transform. The transform space is scanned by an operator adapted from Skingley and Rye (1987). The transform space is scanned until a suspected peak is found which must be the largest value within a 5 by 5 mask. When such a peak is found, eight means (μ_i) and standard deviations (σ_i) of the pixels around the edge of a 13x13 window are calculated (figure 5.2). Skingley and Rye use the maximum mean (μ_m) and standard deviation (σ_m) to calculate a threshold, T, where:

$$T = \mu_m + \alpha\sigma_m$$

where α is the false alarm rate

If the height of the suspected peak is greater than this threshold then the ρ and θ coordinates of the peak give the parameters of a line in the image which is then plotted. This is similar to setting the value of the accumulator to 1 wherever a peak's value is

above the threshold and 0 elsewhere and then performing an inverse transform. Instead of using a binary representation for the accumulator array, the gray levels can be retained. When the peak value exceeds the threshold, it is replaced by the maximum value (i.e. 255), otherwise the transform space is median filtered with a 7x7 mask. The image is then reconstructed using the inverse Radon transform. This image consists only of lines, so the original image is added to produce an enhanced image. In the enhanced image the lines appear brighter

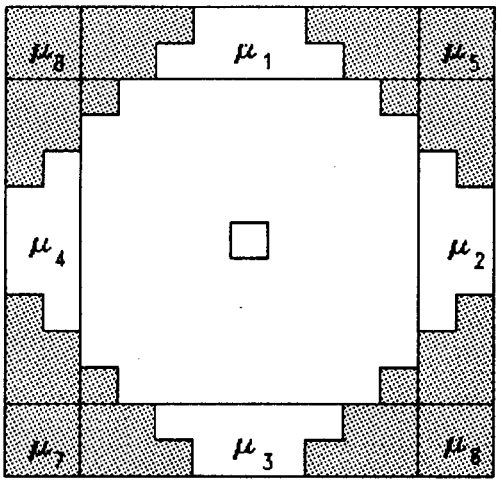
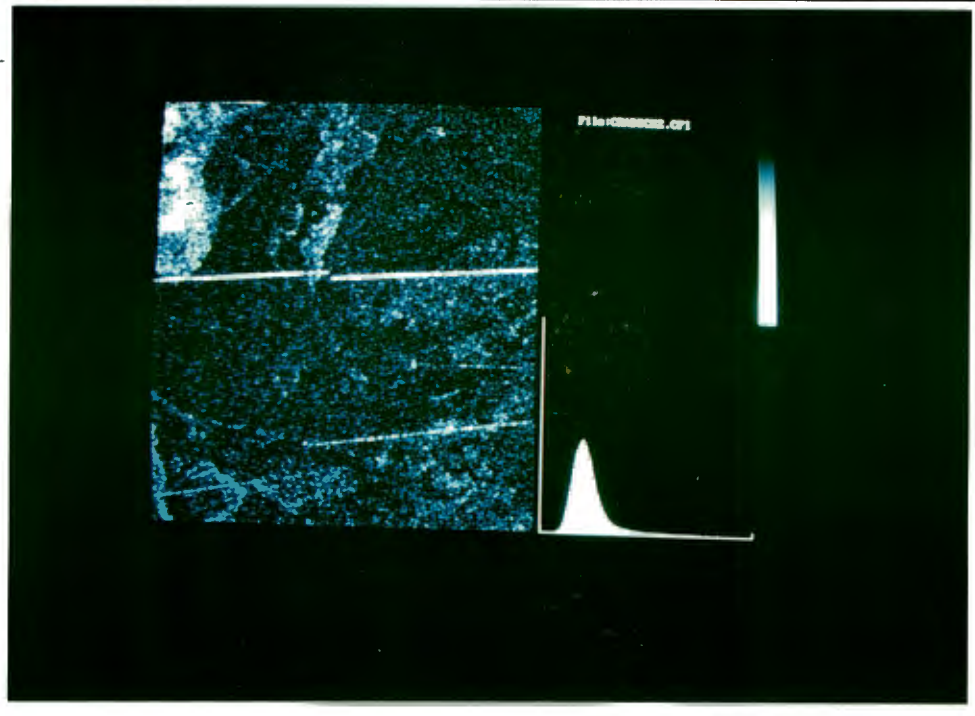
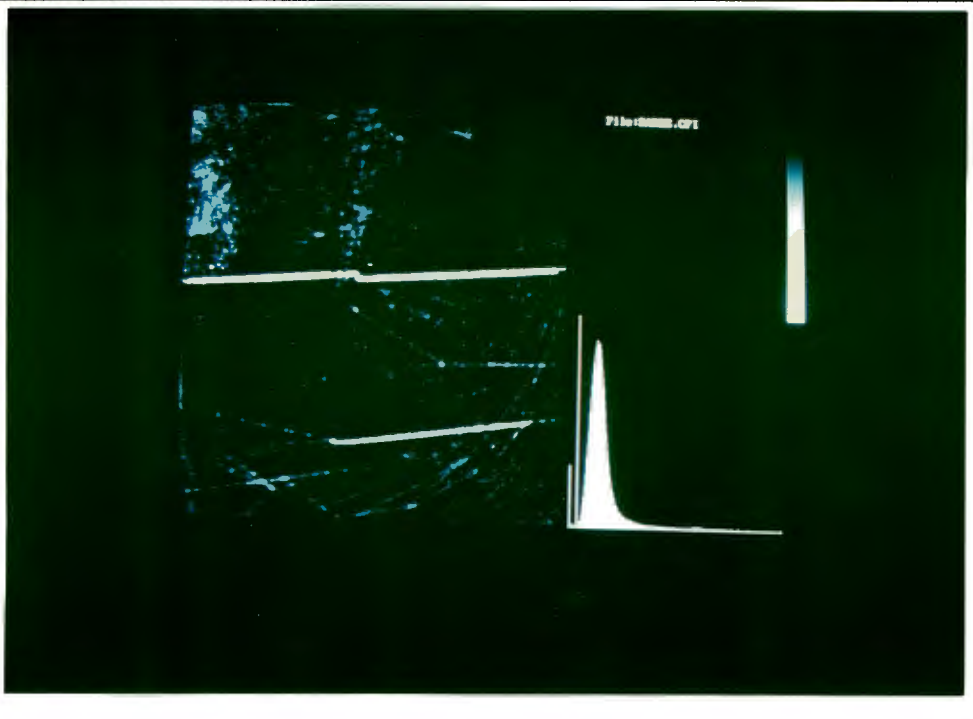


Figure 5.2. Peak detector used for enhancing lines in an image. The mean (μ) and standard deviation (σ) are calculated for each of the eight regions indicated in the diagram. A threshold is then calculated from these values. If the pixel value at the center of the mask exceeds the threshold then it is a peak.

Another example would be the removal of noise from a line segment. If the line in the image is continuous, the butterfly will be smooth. If not, the butterfly will have 'streaks'. Therefore to make a noisy line (with missing points) continuous it is necessary to remove the 'streaks' from the butterfly. This can be done with simple low pass filtering (Dachs and De Jager, 1990). However, the streaks widen as distance from the peak increases. Most of the energy in the butterfly is contained near the peak and so it is in this area that the filter must be most effective. It is unimportant if some streaks remain in areas far from the peak as the contribution from these areas is small. Even so, a fairly large area of support is needed for the filter. The size of the area of support needed can be found by considering the following argument.



(a)



(b)

Figure 5.3. Synthetic Aperture Radar image of Grahamstown area. (a) Original image. (b) Enhanced version. The lines are enhanced using the Radon transform/ filter/ inverse Radon transform technique. 50% of the original image is added to preserve those parts of the image that are not linear features.

The Hough transform of a single line generates a single peak in Hough space. Thresholding the Hough space at such a level that only the top of the peak remainsgenerates an infinitely long straight line when the image is reconstructed. Taking all of the information into account regenerates the line exactly (apart from the reconstruction artifacts). In between these two extremes there must be a threshold level for which the line can still be reconstructed. The width of streaks at this level will determine the area of support needed. Figure 5.4 shows the result of thresholding the Hough space at various levels.

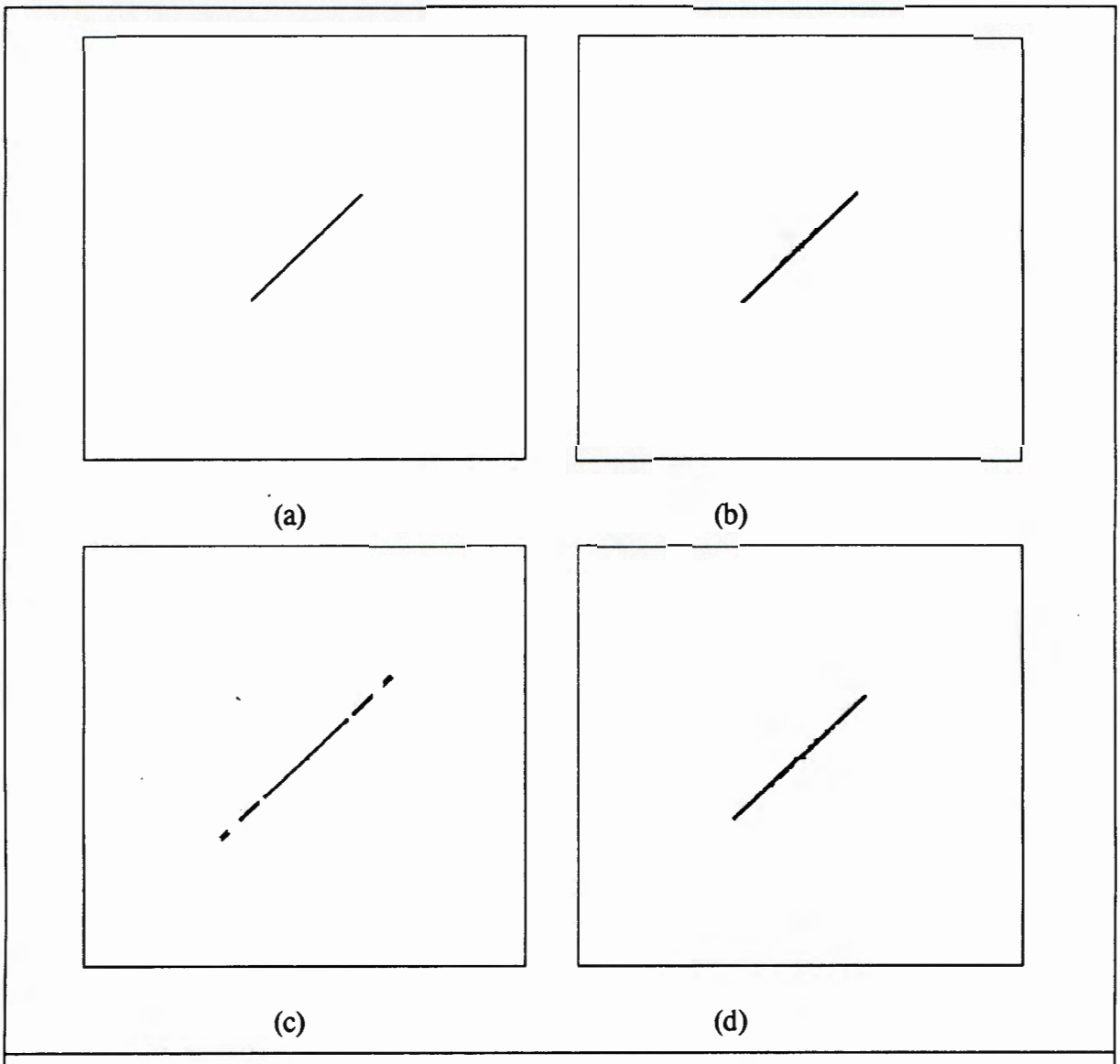
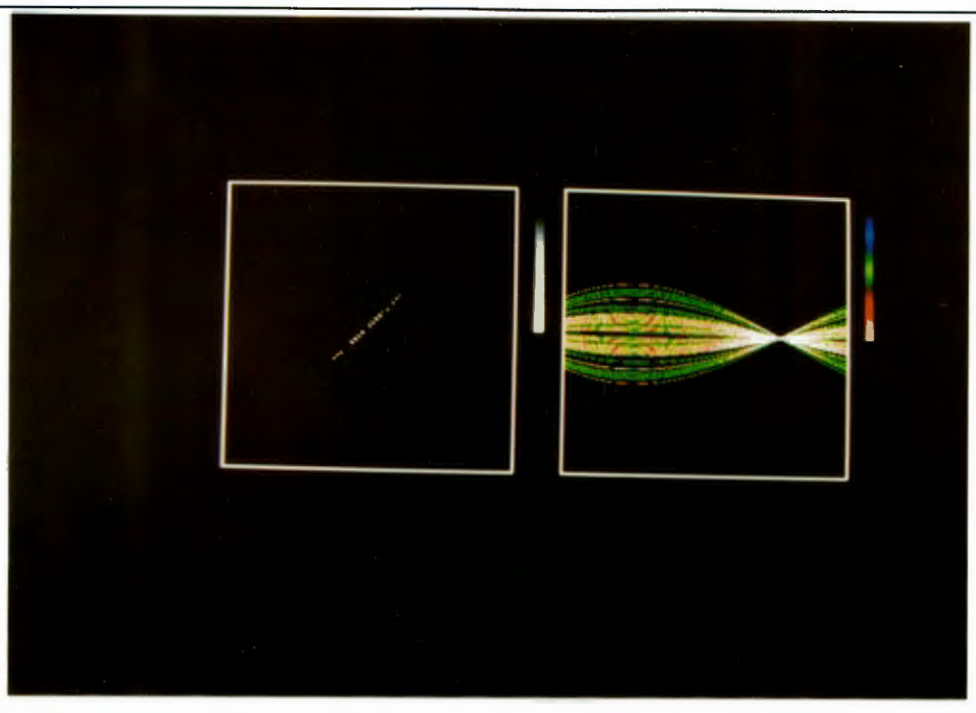
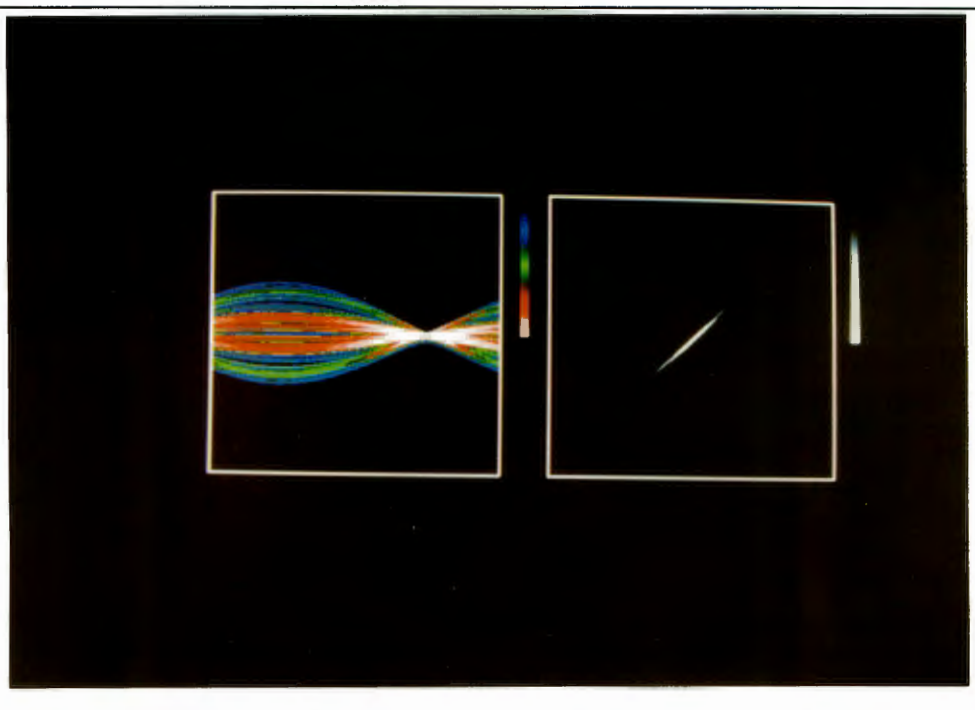


Figure 5.4. Reconstructed lines with varying threshold levels. (a) Original line. (b) Reconstructed line from full Hough information. (c) Reconstructed line when Hough space is truncated below 6%. (d) Reconstructed line when Hough space truncated below 1.5%.



(a) (b)



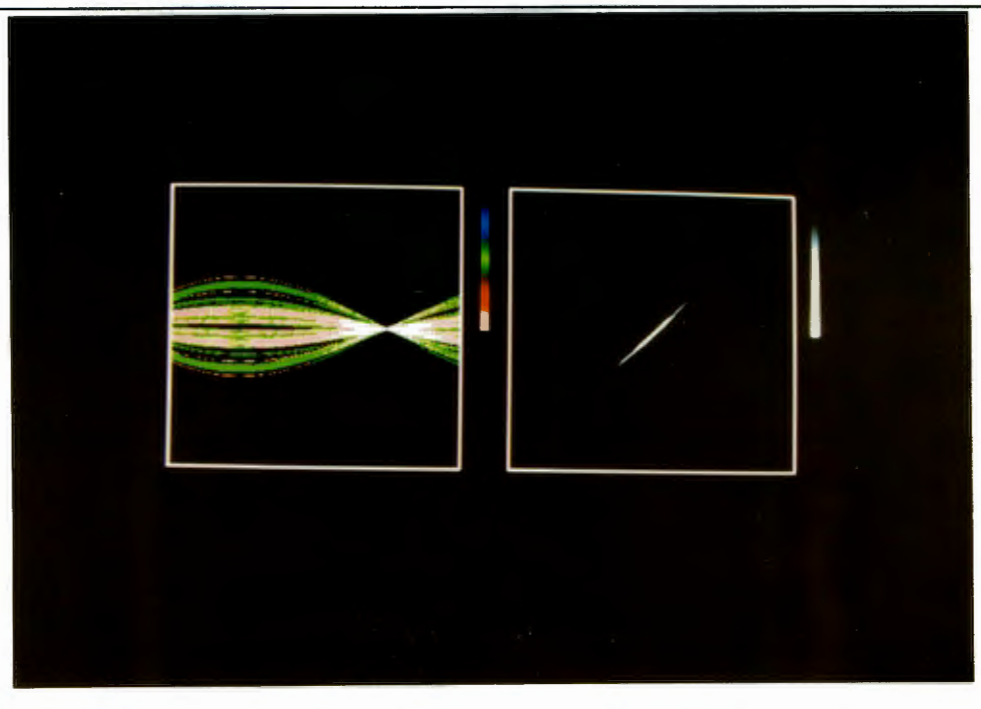
(c) (d)

Figure 5.5. The four stages for a noisy line segment. (a) Noisy line segment. (b) Butterfly resulting from noisy line segment. (c) Smoothed butterfly. (d) Result after performing inverse Radon transform.

As can be seen, the line is only reconstructed accurately when the threshold is set to a low value. This means that the shape of the butterfly is important for a relatively large distance from the peak. Therefore a large filter mask will be needed for smoothing.

Figure 5.5 shows the stages for smoothing a line. The Hough transform was performed on the binary image. The resulting transform was smoothed using a 7x7 convolution mask (all of the elements are 1, and the result is scaled by 1/49 to give unity gain). The transform was then inverted with the inverse Radon transform. The method used to reconstruct the image was the projection slice theorem.

The lowpass filtering action in the above example thickens the line. Instead of a lowpass filter a median filter could be used. A median filter, however, will eliminate any spikes. The peak of the butterfly is such a spike, and would therefore, be eliminated. To remedy this a modified median filter is used. The median filter only replaces the centre value by the median if the range of values is within a preset limit. The limit is set so that the peak is larger than this limit, therefore the filter does not eliminate it.



(a) (b)

Figure 5.6. Smoothing of a noisy line segment using median filtering. (a) Median filtered butterfly. (b) Reconstructed image.

5.2 Image Analysis

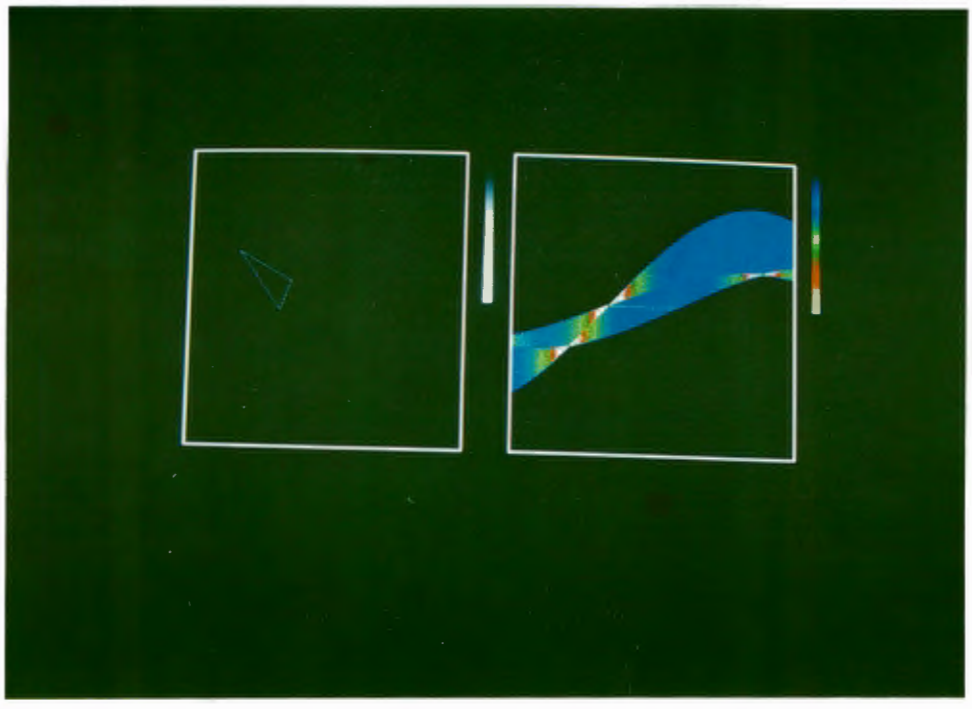
In image analysis the aim is to extract information that can be passed on to a machine. Therefore, the output does not necessarily have to be in the form of an image. For convenience the results shown here are in the form of images.

One problem that arises when using the Hough transform to extract information about 3D scenes is that the end points of lines in the image are unknown. This problem was discussed in an earlier chapter. Essentially, intersections can be found that do not actually exist because the start and end points of each line are unknown.

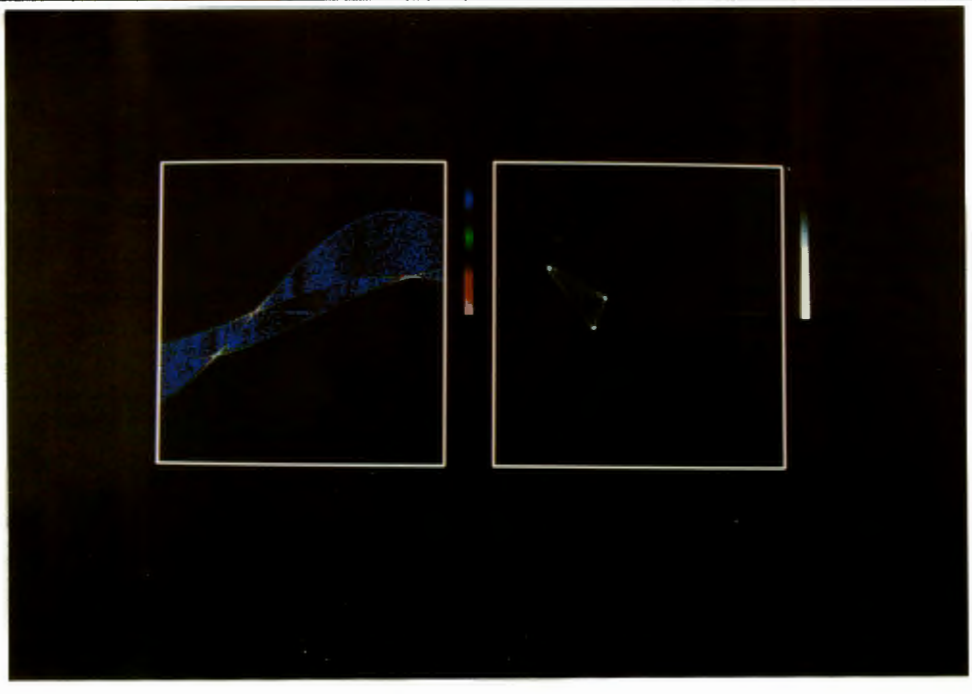
To solve this problem, another parameter of the butterfly shape can be used to calculate the end points of the line segment. It can be shown geometrically that the extreme edges of the butterfly are generated by the end points of the image line. Therefore by filtering the Hough space, to remove all but the edges of the butterfly, the information about the position of the end points may be found. The resulting space can be inverted to produce an image consisting only of points. These points will be the end points of any lines in the original image (Dachs and De Jager, 1990). Figure 5.7 shows these results on a synthetic image.

Figure 5.8 shows the same operations on a real image. A video camera was used to obtain the image of a pyramid. Here an edge detector is used to pick out the edges before applying the Hough transform. The end points of the lines are shown as bright points in figure 5.8 (d), and in this case correspond to vertices of the pyramid.

This technique gives the start and end positions of any line in the image, and as such completely describes the line segment. However, when there is more than one line segment in the image the start and end points must be paired together for each particular line. With the information from the peak position we also have the ρ, θ parameters of each line segment. This information can be used to perform the matching.

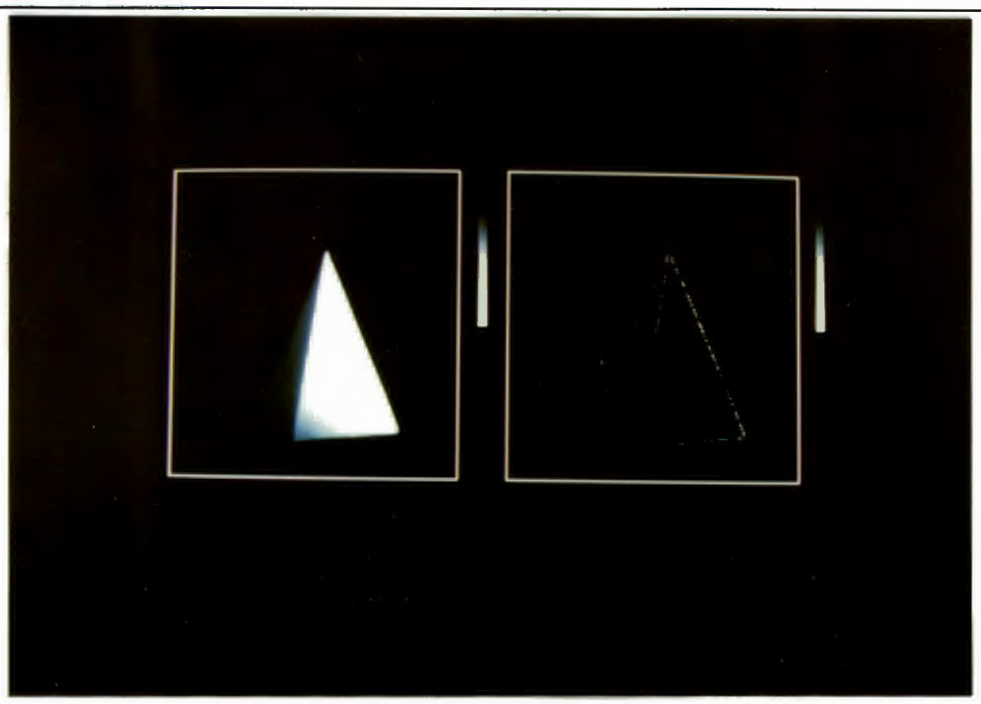


(a) (b)

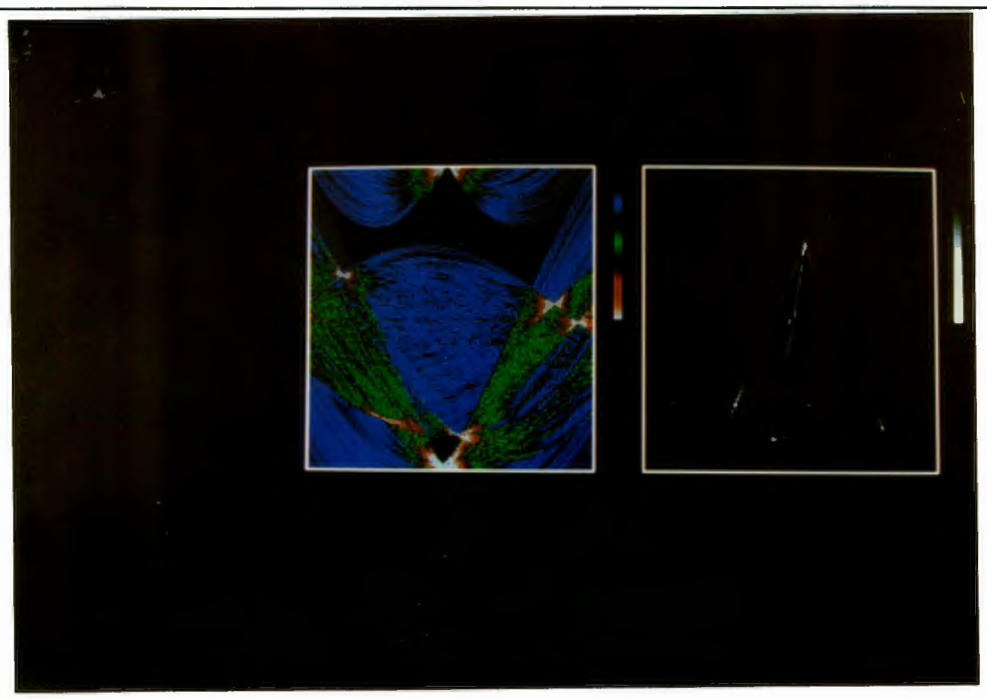


(c) (d)

Figure 5.7. Finding the end points of line in a synthetic image. (a) A binary triangle. (b) Hough transform of the triangle. (c) Result of edge detection on the Hough transform. (d) Inverse showing the end points. The end points can be extracted using a threshold.



(a) (b)



(c) (d)

Figure 5.8. Finding the end points of line in a real image. (a) Pyramid. (b) Edges extracted from pyramid image. (c) Hough transform of edges. (d) Inverse showing the end points.

5.3 Analysis of scenes containing Polyhedra

In many applications, such as robot vision, it is necessary to extract a connected line drawing from a perspective view. Such line drawings could be the input to Computer Aided Design (CAD) systems which model three dimensional objects (De Floriani, 1989). The description typically required by such systems is the position of the vertices.

It is possible to develop algorithms that are applicable to any type of scene, but these tend to be difficult to use for extracting geometrical and topological properties (Gu and Huang, 1985). The class of possible scenes is limited to those containing polyhedra in order to use *a priori* information. This is still a useful class of scenes in industrial applications. If only polyhedra are being considered the following observation can be made. Namely, the image consists of straight line segments and these segments join corners of the polyhedron. This is, however, only true for an image of a single polyhedron.

Gu and Huang (1985) developed an algorithm to generate such drawings. Their approach was to use a Sobel edge detector to find the edges in the image, then to use the gradient information from the detector to find the corner points. The line positions are found by assuming that between each pair of corner points there is a line segment. The direction of the line segment is found from the gradient direction in a local region around each corner. The remainder of their algorithm sifts through the list of corner points to eliminate false corners and lines. They do not appear to take into account the possibility that there may be more than one polyhedron in the image. This may result in genuine corner points that do not actually have lines joining them.

Wahl and Biland (1986) used the Hough transform to extract primitive polyhedrons from images. The Hough transform is a robust process for finding the position of lines. However, as pointed out in chapter 2, it does not give the position of the end points of the line. One difficulty encountered was the accidental aligning of independent vertices in the 2D projection. This effect could be reduced by using the position of the end points of the lines.

Wu and Rosenfeld (1983) used filtered projections as an aid to corner detection. Here, two projections were taken in orthogonal directions, for example, in lines parallel to the x and y axes. The slope discontinuities in the projected data are enhanced by applying a convolution operator. An example of the type of operator that could be used is a Laplacian.

Peaks in each projection give the positions of corners. Additionally, it was shown that the process of projecting the data reduces zero mean random noise.

The method proposed here has features common to all three of the above methods. The Hough transform (which is a set of projections) is used to find the position of the lines, giving the ρ and Θ parameters of each line. Additionally the Hough transform is used to find the positions of the ends of each line segment. This is done using the method given earlier in section 4.2.2. This method can be considered to be an extension to Wu and Rosenfeld's method. Instead of considering only two projections, a large number are used. Also, instead of a one dimensional Laplacian operator, the two dimensional Sobel operator is used.

The inverse Radon transform is then used to invert the transform, resulting in peaks at the end-point positions. A peak finding algorithm is used to find the position of each of the end points to within an accuracy of one pixel.

Following this, two lists are created. The first is a list of the lines in the image (each line is assumed to be infinitely long). The second is a list of the end points.

The ρ and Θ parameters of a hypothetical line joining each pair of points in the second list are calculated. If these ρ, Θ parameters match those for a line in the first list then a line might exist between these two points. A third criterion is used to decide whether such a line does actually exist. This is necessary because there may be more than two collinear end points. The density of the input image is measured between the two estimated end points by counting the number of pixels on an edge and dividing by the length of the line. If this density exceeds a preset threshold then the line is added to the list of lines in the image. The final description is a list of lines giving the start and end point for each line.

This method is not limited to a single polyhedron, nor is it limited to polyhedra. In fact, any image containing line segments can be analysed.

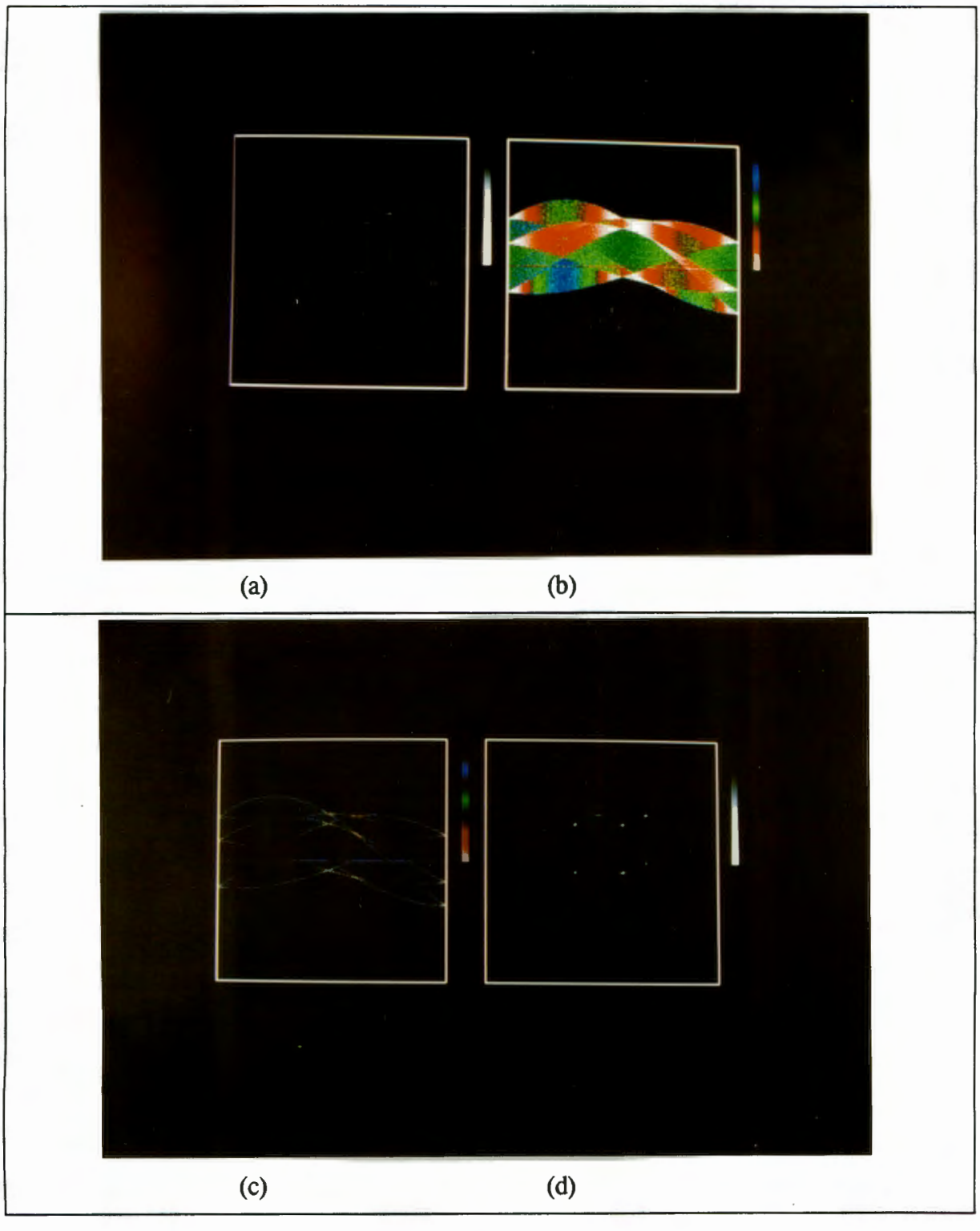
The complete algorithm is:

1. Find the edges in the scene
2. Perform the Hough transform on the edges
3. Find the edges in Hough space
4. Perform inverse Radon transform
5. Find peaks in the inverse (end points of lines)
6. For each pair of points
    6.1 Check for a peak in Hough space that has parameters close to those for a line joining the two points
    6.2 If such a peak exists then check that the density of the image between these two points is greater than a preset level
    6.3 If this is also true then add a line to the final list with these end points

The process is shown in figure 5.9 for a synthetic image. The original image is in edge form already so the Hough transform can be done directly. The Hough space was filtered using a Sobel edge detector. This was then inverted using the Filtered Back Projection algorithm. The result is shown in figure 5.9(d). A peak finding algorithm was used to find the coordinates of each end point. These coordinates were matched with the line information from the Hough transform to give the final result in figure 5.9(e). The green points are those found by the end-point-finding algorithm. The red lines are the lines found in the Hough transform that may fit point pairs. The line density along such lines is used to decide whether the point pairs should be joined. The white lines are the final lines.

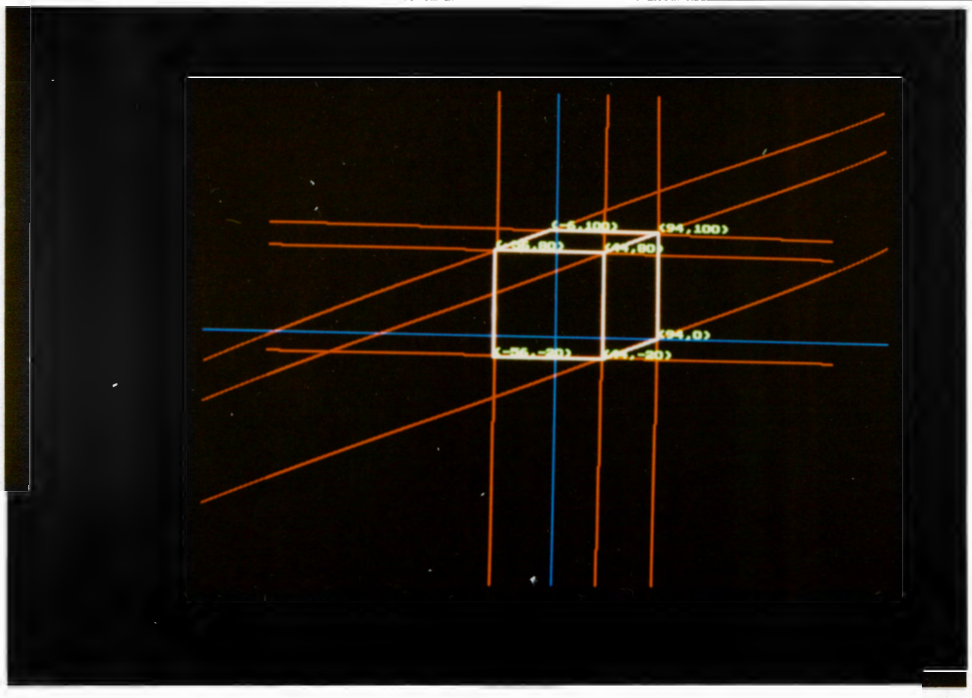
Figure 5.10 shows the same process for a real image. The image was taken using a home video camera. It was illuminated with many diffuse light sources. The image required median filtering before processing with the edge detector. A 3x3 median filter was used for the filtering. The valley detector was used in preference to the Sobel edge detector. It was found that the thresholded output from the Sobel edge detector was unsuitable because the strong edges (around the outside of the polygon) generated thick lines while the faint edges generated thin lines. The valley operator discussed in chapter 2 gave thin lines for both type of edge (See appendix A for a description of the median filter and edge detection programs).

However, the output of the Hough transform was processed using the Sobel edge detector. The Hough transform was inverted using the filtered back projection algorithm and the end points found using the same peak finding algorithm as before.



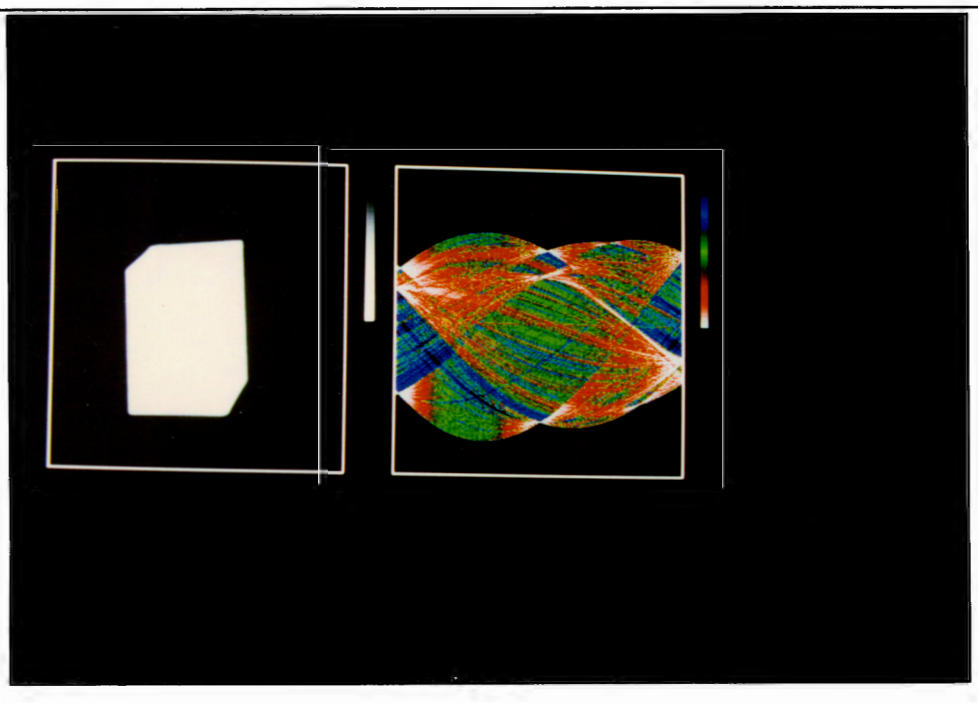
(a) (b)

(c) (d)

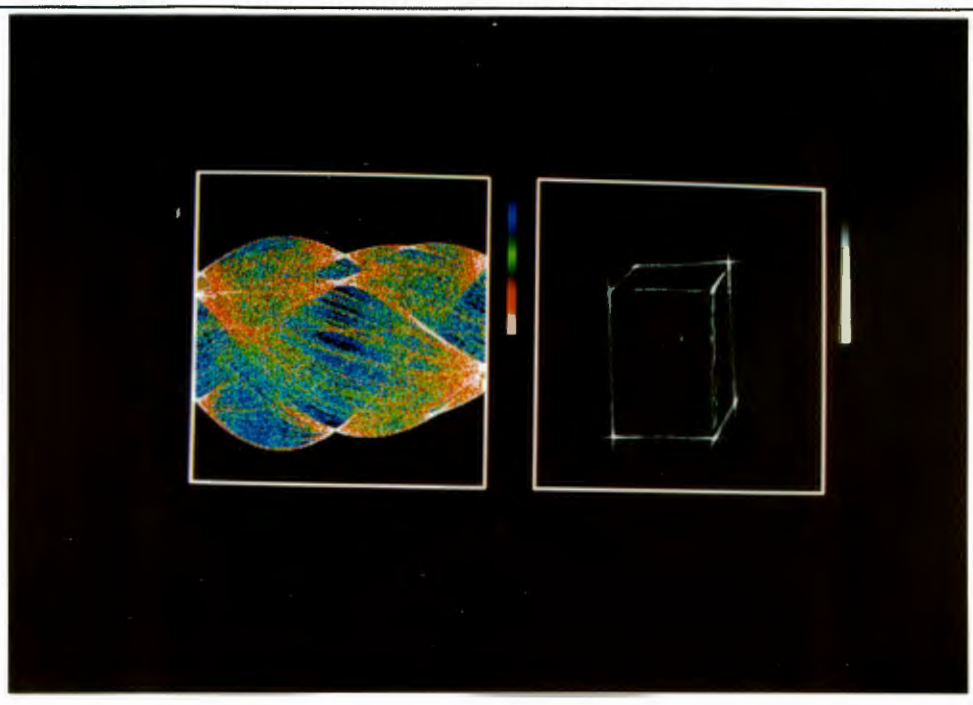


(e)

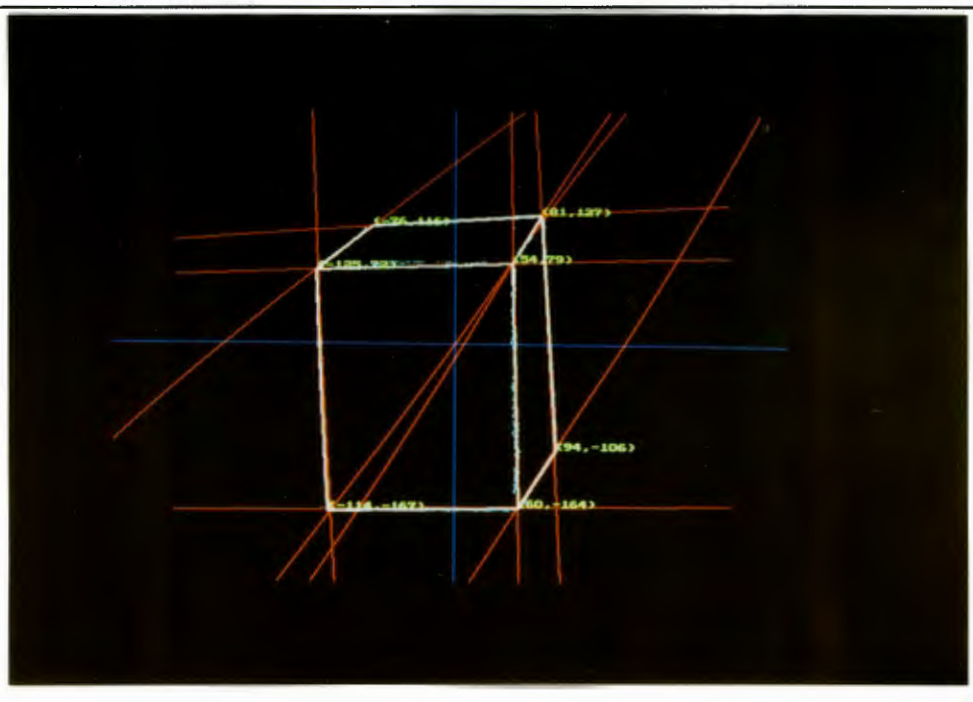
Figure 5.9. Analysis of polyhedral scenes. (a) Original image. (b) Hough transform. (c) Filtered Hough transform. (d) End points found by inverting the filtered transform. (e) Matched points and lines.



(a) (b)



(c) (d)



(e)

Figure 5.10. Analysis of a real polyhedral scene. (a) Original image. (b) Hough transform of edges in (a). (c) Filtered Hough transform. (d) End points found by inverting the transform. (e) Matched points and lines. The red lines are those found by the Hough transform. The blue lines are the edges found by the edge detector. The corner coordinates are found by the end point finding algorithm and the white lines are result of analysing the image.

The major difficulty encountered when applying this method to real images was the generation of an adequate edge image. The edge detectors found the images to be noisy and that some of the edges were extremely faint. The Sobel edge detector generated edges of different thicknesses. The output from a Sobel edge detector can be modified to give thin edges using the method in Gu and Huang (1985). In this case an edge element is said to exist if the magnitude of the central pixel exceeds a preset value and it also exceeds the magnitude of its neighbours in a direction normal to the direction of the edge. This gives thinned edge points so the results are better than simply taking the magnitude of the Sobel edge detector.

Another difficulty encountered was that it was necessary to set thresholds manually. These thresholds are dependant on the image, but their settings are not critical. An automatic threshold selection method, such as the moment preserving method, could be used.

5.4 Findings peaks in the Hough space

One of the major difficulties encountered when attempting to implement the Hough transform is finding the peaks in the Hough accumulator. The most common method is to fix a global threshold. Any bins exceeding this threshold indicate the presence of a line. However, the height of the peak generated by a line is proportional to the length of the line. As a result, a global threshold is not suitable for images containing lines of different lengths.

The converging squares algorithm can be used to find the peaks (O'Gorman and Sanderson, 1984). This algorithm is computationally efficient and is robust with respect to noisy data. However, it was found that spurious peaks were detected, even in the absence of noise in the original image, unless the Hough space was filtered by a low pass filter before applying the converging squares algorithm.

Skingley and Rye (1987) proposed a peak finding algorithm which they used to detect faint lines in SAR images (see section 4.1). However, this method requires a 'false alarm rate' to be set and appears to be fairly sensitive to this setting. False lines can be eliminated by measuring the image density in the original image. If the density is lower than a threshold, then the candidate line is false.

Several authors have proposed iterative schemes. In this case the highest peak in the Hough space is found. This corresponds to a feature in the image. The contribution from this feature is subtracted from the Hough space. The process is repeated until there is no peak above an arbitrary threshold. A difficulty arises when attempting to estimate the contribution of each feature to the Hough space. In the case of the straight line transform, only two parameters of line are known from the peak position, so the line is assumed to be infinitely long. The contribution from an infinitely long line will be different from the contribution from a short line segment. O'Gorman and Clowes (1976) solved this problem by keeping a list, for each cell in the accumulator, of the image points contributing to the cell. When a peak is detected the lists are scanned for counts arising from the same feature. This method appears to be reliable, but requires an extremely large amount of storage and processing.

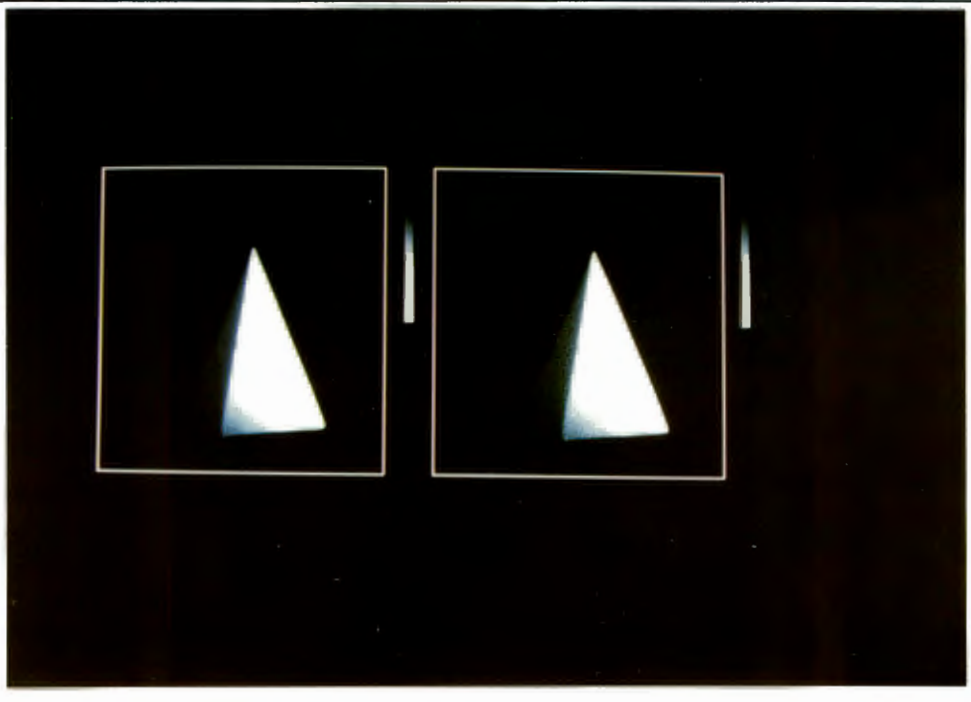
The method described in 4.2 gives the end points of each line segment. Therefore, when a peak is found in the Hough accumulator, the correct cells can be decremented. This method is also computationally intensive because it requires the computation of the inverse Radon transform. In practice it was found that errors, particularly in the early stages, propagate. If the first line is not detected correctly, incorrect cells in the Hough accumulator are detected. This, in turn, leads to errors when detecting subsequent lines.

5.5 Noise performance

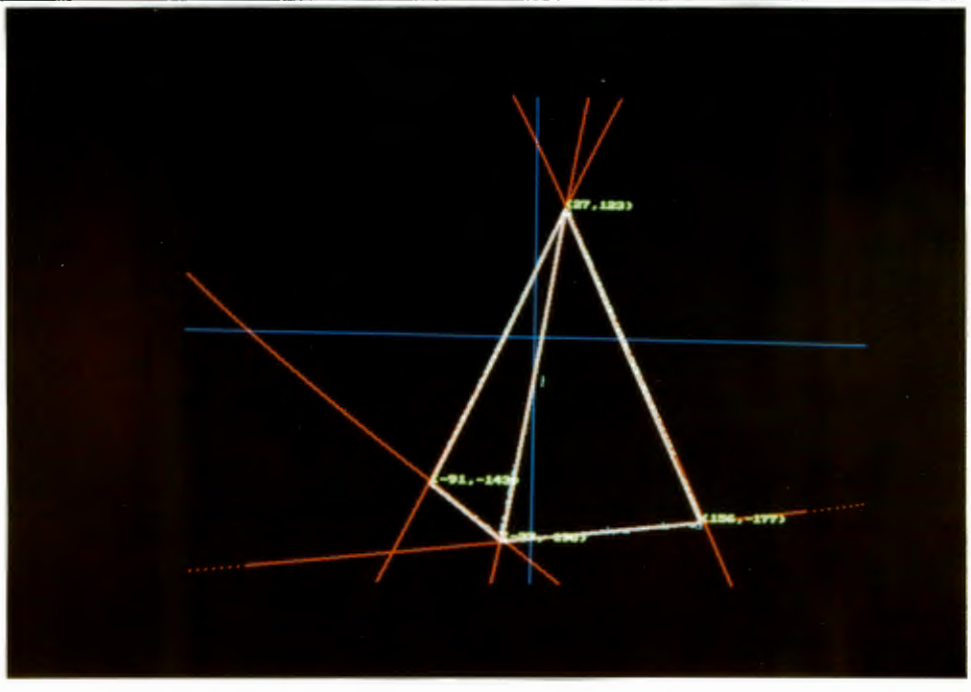
To evaluate the performance of the polyhedral analysis algorithm, a known amount of Additive White Gaussian Noise (AWGN) is added to an ideal image. The process of analysing a polyhedral scene is then applied to the image. Therefore, the signal-to-noise ratio that causes the method to break down can be found.

The image used is that of a pyramid (see figure 5.11(a)). The image was taken with a video camera so it already has a certain amount of noise present. The signal to noise ratio (SNR) can be measured using the technique proposed by Janse Van Rensburg (Janse Van Rensburg and De Jager, 1990) and was found to 40 dB. AWGN was added to this image and the signal to noise ratio was degraded to 24.7 dB (figure 5.11(b)). The polyhedral analysis procedure works correctly on this image as shown in figure 5.11(c). AWGN is

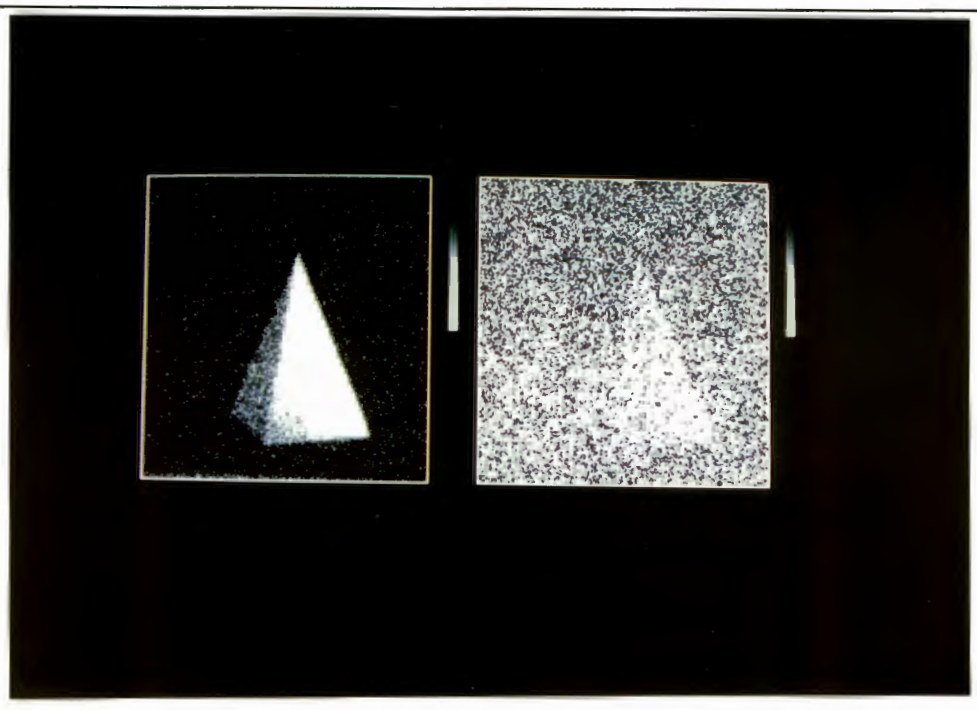
added in increments, finally decreasing the SNR to 4.6 dB (figures 5.11 (d) and (e)). At approximately 20 dB SNR the analysis program becomes unreliable.



(a) (b)



(c)



(d) (e)

Figure 5.11. Failure of polyhedral analysis method in the presence of AWGN.
(a) Original image (SNR = 40 dB). (b) AWGN noise added (SNR = 24.7 dB).
(c) Analysed image. (d) Image with SNR = 14.6 dB. (e) Image with SNR = 4.6 dB.

CHAPTER 6

6. Suggestions for Further Research

The results, so far, have been presented for the straight line version of the Hough transform. In chapter 2 it was shown that the Hough transform could be extended to other shapes. Chapter 3 showed that the Radon transform was similar to the Hough transform for ρ,θ parameters. Since the Radon transform is invertible the inverse can be used to invert the straight line Hough transform.

The other versions of the Hough transform have no such inverse. This chapter will attempt to find inverse transforms for these other Hough transforms. The approach will be to use the ρ,θ case as a model to develop theoretical solutions for other parameterisations.

6.1 Straight line Model

The Radon transform of a function $f(x,y)$ is (Deans, 1983):

$$P_\theta(t) = \int_L f(x,y)\ ds$$

where L is a line in the image and ds is an incremental distance along this line.

An alternative form of this expression is:

$$P_\theta(t) = \iint f(x,y)\cdot\delta(t - x\cdot\cos\theta - y\cdot\sin\theta)\ dx\ dy$$

where the equation of the line L, in this case, is:

$$t = x\cdot\cos\theta + y\cdot\sin\theta$$

This expression evaluates to zero on the line L so the double integral in the above expression is evaluated only for points on the line L.

The filtered back projection algorithm is a means for inverting this transform. The equation can be written as (as shown in chapter 2):

$$f(x,y) = \int_0^{\pi} Q_\theta(x \cdot \cos\theta + y \cdot \sin\theta)\, d\theta$$

and since $t = x \cdot \cos\theta + y \cdot \sin\theta$

$$f(x,y) = \int_0^{\pi} Q_\theta(t)\, d\theta$$

Here $Q_\theta(t)$ is the projection, $P_\theta(t)$, filtered by a filter which has the response $|\omega|$ in the frequency domain. The effect of filtering a signal by such a filter is to differentiate the signal. Therefore:

$$Q_\theta(t) = \frac{d}{dt}(P_\theta(t))$$

and:

$$f(x,y) = \int_0^{\pi} \frac{d}{dt}(P_\theta(t))\, d\theta$$

Thus, in the Hough parameter space, which has t as one axis and θ as the other, the above corresponds to differentiation in one direction and integration in the orthogonal direction. This technique is also used by edge detectors to reduce the effect of noise. This noise reduction may explain the greater popularity of the filtered back projection method, as compared to using the projection slice theorem.

6.2 Conic Sections

Conic sections include circles, ellipses, hyperbolae and parabolas. These curves represent features arising frequently in images. Ellipses occur particularly frequently since circles, viewed at an oblique angle, becomes ellipses in the image. However, for simplicity, the circle is used here. The Radon transform (for straight lines) is:

$$P_\theta(t) = \int_L f(x,y)\ ds$$

It is proposed that this can be adapted to the circular case by replacing L by the equation of a circle instead of a straight line. The form of the equation is:

$$x^2 + y^2 = r^2$$

or:

$$r^2 - x^2 - y^2 = 0$$

and ds is an incremental distance along this circle. Therefore:

$$P_\theta(t) = \iint f(x,y) \cdot \delta(r^2 - x^2 - y^2)\ dx\ dy$$

Writing this as an algorithm for a digital image gives the following:

```
1. For each x,y point in the image
     Find r = √(x² + y²)
     A(x,y,r) = A(x,y,r) + f(x,y)
```

This will generate a three dimensional accumulator array with peaks at `(x,y,r)` giving the parameters of any circles in the image and is equivalent to the Hough transform for finding circles described in chapter 2. This is also the equivalent to the Radon transform for straight lines. However, instead of projecting along straight lines, the projections are taken along circular contours.

To find an inverse transform for this version of the Hough transform requires the inversion of the integral given above. The proofs for the two methods for inverting the Radon transform (the filtered back projection and projection slice theorem) depend on the fact that L is a straight line of the form:

$$t = x \cdot \cos\theta + y \cdot \sin\theta$$

These proofs are invalid if the path of integration is not a straight line with the above form. The mathematical proof is beyond the scope of this study, however, it is proposed that the integral should be invertible for other functions.

If the integral is invertible, then there is a forward/inverse transform pair for conic sections. In the case of a circle transform, this transform pair would enable arcs to be found.

The computational requirements for Hough transforms of any shape other than a straight line are large. Usually the process is divided into several sequential stages. For example, the centres of candidate circles are found in the first stage, and then their radii estimated in a second. This reduces the storage and computing requirements.

This method is typically used for finding ellipses, since ellipses are described by five parameters. The method used by Muammar and Nixon (1990) has three stages. The first is to estimate the centres of the ellpses. The second estimates their orientation and the third estimates the major and minor axes. Inverse transforms could be found for each stage, which in turn would allow extraction of information not normally available. Specifically, the first stage produces a parameter space with peaks at the centres of candidate ellipses. If an inverse transform were available for this stage it would be possible to determine which section of the ellipse gave rise to the peak. It would, therefore, also be possible to determine if the ellipse was partially occluded.

6.3 Arbitrary Shapes

In the equation for the Radon transform:

$$P_\theta(t) = \int_L f(x,y)\, ds$$

the equation of the path of integration is given by L. For arbitrary, non-analytic shapes, this equation cannot be written analytically. However, the Hough transform can be performed for such functions by storing a table of radii (the generalised Hough transform described in chapter 2). Since this table describes the outline of the shape, it defines the path of integration. It is proposed that the inverse can also be found for the generalised

Hough transform. However, the computational restrictions mentioned above will also apply to this case.

CHAPTER 7

7. Discussion and Conclusions

The aim of a machine vision system is to provide a description of a scene by extracting information from one or more images. In this dissertation the Hough transform was investigated in this context. The Hough transform is a robust and efficient method for finding straight lines in an image and it was used as a means for extracting information about polyhedral shapes in an image. This is an example of syntactic pattern recognition, where simple features are extracted first. The simple features (the lines) are then used to extract more complicated features (polyhedra). Furthermore, results were presented to show alternative uses of the method presented here.

The basis of the method discussed is that the Hough transform is invertible. The inverse transform is derived from the Radon transform, since the Hough transform has been shown to be a special case of the Radon transform. Techniques for inverting the Radon transform are readily available since the Radon transform is important in medicine and has been well studied. The Hough and inverse Radon transforms form a forward / inverse transform pair. The Hough transform produces an accumulator array, or Hough space. In this space, straight lines are represented by peaks. Therefore the process of finding lines is reduced to the simpler task of finding peaks. Similarly linear features can be enhanced by filtering the peaks in the Hough space. The inverse transform provides the means to reconstruct the image.

The Hough, Radon and inverse Radon transforms were implemented on general purpose computers. The computation times for these transforms, particularly the inverse transform, are long. However, it was found that it is possible to develop dedicated hardware that will perform forward and inverse transforms in near real time.

Several results of the application of the forward transform / filter / inverse transform were given. The method was used in image enhancement to enhance linear features in synthetic aperture radar images. It was also used to smooth noisy lines in computer-generated images. However, convolution masks with a large area of support were required for the filtering stage. This fact, combined with the computations required for the forward and

inverse transforms means that this is not a particularly efficient method for enhancing linear features.

Results were given for image analysis, in particular, finding the end points of line segments in the image. Other methods for finding corners directly in the image have been developed, but they are computationally expensive and in some applications it may be preferable to find end points rather than corners. The method presented here requires filtering the Hough space with an edge enhancement operator to find the end points. It is fairly robust with respect to noise in the image and its sensitivity can reduced by first smoothing the Hough space. This smoothing is one of the techniques discussed in the section on image enhancement.

The line direction information obtained from the position of the peaks in the Hough transform was combined with the line end point information to analyse scenes containing polyhedra. A polyhedron consists of line segments, so with the above information, the position of any polyhedron can be found. The information extracted could be passed on to a geometric modelling program. These programs have already been developed and are used extensively in mechanical engineering and architecture. Such a model has access to a data base of information about the types of polyhedra that can exist and can therefore match the information presented to it, to various models.

The end point information can also be used to find the peaks in the Hough space in a robust manner. As each peak is found, its contribution to the Hough space can be subtracted. The residual can now be inspected for smaller peaks that may have been hidden by larger peaks.

These methods show the applications of the end-point-finding algorithm. This algorithm, works for straight lines. However, the Hough transform has been extended to other parametric curves. The methods presented here cannot be applied to the other versions of the Hough transform directly because there is no inverse transform. A general form for the derivation of such inverses has been suggested.

The objectives of the thesis were fulfilled. The Hough space produced by the straight line Hough transform has been analysed and it has been shown that the end points of line segments can be extracted. It has also been shown that linear features can be enhanced by filtering in Hough space. These methods can be implemented on general purpose

computers, but dedicated hardware will be required for a real time implementation. Such hardware is technically and economically feasible.

List of References

Ballard D. H., Generalizing the Hough Transform to Detect Arbitrary Shapes, *Pattern Recognition*, Vol. 13, No. 2, pp. 111-122, 1981.

Ben-Tzvi ,D. and Sandler, M., Analogue Implementation of Hough Transform, *Electronic Letters*, Vol. 25, No. 18, August 1989.

Besl, Paul J., Geometric Modelling and Computer Vision, *Proceedings of the IEEE*, Vol. 76, No. 8, pp. 936-958, August 1988.

Dachs, Andrew and De Jager, Gerhard, Filtering in Hough Space, *Proceedings of COMSIG 90*, IEEE, pp. 164-169, June 1990.

Davies E.R., Finding ellipses using the generalised Hough transform, *Pattern Recognition Letters*, North Holland, Vol. 9, pp. 87-96, 1989.

De Floriani, Leila, Feature Extraction from Boundary Models of Three-Dimensional Objects, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 8, August 1989.

Deans, S.R., Hough Transform from the Radon Transform, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-3, No. 2, March 1981.

Deans, S.R., *The Radon Transform and Some of its Applications*, John Wiley & Sons, New York, 1983.

Duda R.O. and Hart P.E., Use of the Hough transformation to detect lines and curves in pictures, *Communications of the ACM*, Vol. 15, No. 1, pp. 11-15, January 1972.

Dudgeon, D.E., Mersereau, R.M., *Multidimensional Digital Signal Processing*, Prentice Hall, New Jersey, 1984.

Gu, W.K. and Huang, Thomas S., Connected Line Drawing Extraction from a Perspective View of a Polyhedron, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-7, No. 4, July 1985.

Hanahara, K., Maruyama, T. and Uchiyama, T., A Real-Time Processor for the Hough Transform, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10, No. 1, January 1988.

Hinkle, E.B., Sanz, J.L.C., Jain, A.K., Radon-Based Image Processing in a Parallel Pipeline Architecture, *SPIE Proceedings*, International Society for Optical Engineering, Vol 696, pp 140-145, 1986.

Hough, P.V.C. Method and means for recognizing complex patterns. *U.S. Patent 3,069,654*, Dec. 18, 1962.

Illingworth, J. and Kittler, J., A Survey of the Hough Transform, *Computer Vision Graphics and Image Processing*, Vol. 44, pp. 87-116, 1988.

Janse Van Rensburg, Cornelius and De Jager, Gerhard, Non-Intrusive Measuring of Noise on a Television Picture, *Proceedings of COMSIG 90*, IEEE, pp. 100-103, June 1990.

Kent, Shneier, Eyes for Automatons, *IEEE Spectrum*, March 1986.

Kimme, C., Ballard D. H. and Sklunksy J., Finding circles by an Array of Accumulators, *Communications of the ACM*, vol. 18, no. 2, p. 120-122, February 1975.

Lim, Jae S., *Two Dimensional Signal and Image Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

Mersereau, R.M., Oppenheim, A.V., Digital Reconstruction of Multidimensional Signals from Their Projections, *Proceedings of the IEEE*, Vol. 62, No. 10, October 1974.

Muammar H.K. and Nixon M., A 3-Stage Procedure for Ellipse Extraction, *1990 Research Journal*, Dept. of Electronics and Computer Science, University of Southhampton, 1990.

Murphy, Lesley M., Linear feature detection and enhancement in noisy images via the Radon Transform, *Pattern Recognition Letters* (North Holland), Vol. 4, pp. 279 - 284, 1986.

Nixon M., Application of the Hough Transform to correct for linear variation of background illumination in images, *Pattern Recognition Letters*, Vol. 3, pp. 191-194, 1985.

O'Gorman, Lawrence and Sanderson, Arthur C. The converging Squares Algorithm: An efficient Method for Locating Peaks in Multidimensions, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-6, No. 3, May 1984.

O'Gorman, Frank and Clowes, M. B., Finding Picture Edges Through Collinearity of Feature Points, *IEEE Transactions on Computers*, Vol. C-25, No. 4, April 1976.

Pearson D.E. and Robinson J.A., Visual Communication at very low data rates, *Proceedings of the IEEE*, Vol. 73, No. 4, April 1985.

Poelzleitner, Wolfgang, A Hough Transform Method to Segment Images of Wooden Boards, *8th International Conference on Pattern Recognition*, Paris, IEEE, pp. 262-264, October 1986.

Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T., *Numerical Recipes*, Cambridge University Press, Cambridge, 1986.

Radon, J., On the Determination of Functions from their Integrals Along Certain Manifolds, Translation of Radon's 1917 paper, in Deans, S.R., *The Radon Transform and Some of its Applications*, John Wiley & Sons, New York, 1983.

Rhodes, F. M., Dituri, J. J., Chapman, G. H., Emerson, B. E., Soares, A. M. and Raffel, J. I., A Monolithic Hough Transform Processor Based on Restructurable VLSI, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10, No. 1, January 1988.

Ruther, H., *Personal Communication*, University of Cape Town, 1990.

Rosenfeld A., *Personal Communication*, University of Cape Town, 1990.

Rosenfeld A., Computer Vision: Basic Principles, *Proceedings of the IEEE*, Vol. 76, No. 8, pp. 863-868, August 1988.

Rosenfeld A. and Kak A.C., *Digital Picture Processing*, Volume 1., Second edition, Academic Press, Orlando, 1982.

Rosenfeld A. and Kak A.C., *Digital Picture Processing*, Volume 2., Second edition, Academic Press, Orlando, 1982.

Rosenfeld A., "Coordinate conversion", in *Picture Processing by Computer*, chapter 8.4, pp. 151, Academic Press, 1969.

Skingley, Jacqui and Rye, A. J., The Hough Transform applied to SAR images fo Thin Line Detection, *Pattern Recognition Letters*, North-Holland, Vol. 6, pp. 61-67, June 1987.

Tsuji, Saburo and Matsumoto, Fumio, Detection of Ellipses by a Modified Hough Transformation, *IEEE Transactions on Computers*, Vol. C-27, No. 8, August 1978.

Wahl, Freidrich M. and Biland, Hans-Peter, Decomposition of Polyhedral Scenes in Hough Space, *8th International Conference on Pattern Recognition*, Paris, IEEE, pp. 78-84, October 1986.

Wu, Zhong-Quan and Rosenfeld, Azriel, Filtered Projections as an aid in Corner Detection, *Pattern Recognition*, Vol. 16, No. 1, pp. 31-38, 1983.

APPENDIX A

Table of Contents

1. Introduction

This file describes an image processing suite of programs developed in the Electrical Engineering Department at the University of Cape Town. The idea behind the project is three fold. First it provides a set of commonly required programs for simple image processing tasks. Secondly it defines file formats so that images will be transportable between different frame grabbers, image processing programs and researchers. The third aim is to provide a set of open procedures and functions which make it easier to write specialised programs which remain compatible.

The programs and routines are all written in Turbo Pascal Version 5.5. They are compatible with Turbo Pascal 4.0 and upwards. Assembler routines for speed critical operations may be linked into a users program and such an example is given. Additional programs may be written in any other desired language, but will not be able use the PCGEN unit. However, if they use the same file standards set out in this document they will interface with this suite of programs.

The hardware required is at least an IBM PC with a Hercules Graphics adaptor and 512K of memory. A hard disk is highly recommended, as is 640K memory. To be able to view gray level images an EGA or VGA display is needed. (Alternatively a frame grabber/monitor combination may be used). Due to the nature of images a large amount of information must be manipulated, therefore a fast processor (such as an AT) is recommended. Most computations on images can be reduced to integer operations so a maths coprocessor will only improve performance in a few instances.

The first section of this document describes how to use the image processing programs as they stand at present. Most of the commonly used algorithm's in image processing are already implemented. The second section describes the programmer's interface. This is to allow you to write your programs, while taking advantage of the routines that have already been written.

2. User's Guide

To start using the programs make sure that the current directory is the one where they are installed, then type:

PC-MENU [Enter]

This displays a list of programs. Use the cursor keys to move the highlighted bar to the program you wish to run, then press enter. Press [F1] for context sensitive help.

2.1 Available programs

The program PC-MENU displays a list of all available programs and allows the user to execute any of them.

The following is a list of programs:

Program Name	Input	Output	Description
PC-IMAGE	BIN	Screen	Displays binary file on any type of monitor.
PC-SCROLL	BIN	Screen	Displays binary image on a Hercules monitor and allows user to scroll around image.
PC-BSCROLL	DBN	Screen	Displays double size binary image on a Hercules monitor and allows user to scroll around image.
PC-VIEW2	CFI	Screen	Display gray levels in 16 shades on monochrome VGA monitor.

Program Name	Input	Output	Description
PC-VIEW5	CFI	Screen	Display gray levels in 16/256 levels on a colour VGA monitor in a variety of resolutions.
PC-PRINT	BIN	Printer	Print binary image on dot matrix type printer (See PC-SETUP).
PC-SETUP	None	None	Selects printer driver for PC-PRINT.
PC-INK	BIN	Printer	Print binary image on HP DeskJet / LaserJet.
PC-BPRINT	DBN	Printer	Print double size binary file on dot matrix printer (see PC-SETUP).
PC-BINK	DBN	Printer	Print double size binary file on HP printer.
PC-LIST	None	Screen	Lists all picture files in a given directory (*.CFI, *.BIN, *.PFI, *.TXT).
PC-THRESHOLD	CFI	BIN	Threshold a gray level file at a level set by the user.
PC-HISTOGRAM	CFI	Screen	Display a histogram of the gray level file.
PC-HISTEQU	CFI	CFI	Modify the histogram of the input file to give a flat histogram.
PC-GTOI	CFI	BIN	Produces dithered binary image.
PC-GTOBI	CFI	DBN	Produces double size dithered binary image.

Program Name	Input	Output	Description
PC-FILTER	CFI	CFI	Convolve input image with any 3x3 mask (*.MSK).
PC-FILT3	CFI	CFI	As PC-FILTER, but uses assembler routines for faster execution.
PC-SOBEL	CFI	CFI	Apply Sobel masks to image. Uses assembler routines.
PC-SMOOTH	CFI	CFI	Apply mask 1/9(1,1,1,1,1,1,1,1,1) to image.
PC-EDGE	CFI	BIN	Applys valley operator to image and stores thresholded result.
PC-STRETCH	CFI	CFI	Stretch dynamic range of image to maximum.
PC-HOUGH	BIN	CFI	Calculate rho,theta Hough transform of input image.
PC-DRAW	None	BIN	User draws image on Hercules screen which is then saved to a binary file.
PC-CONVERT	Any	Any	Converts between the various file formats.
PC-FFT	CFI	CFI	Two dimensional FFT.
PC-FFTT	CFI	CFI	As PC-FFT, but faster.

2.2 Description of each program

Each program is described breifly below. The programs are listed in alphabetical order.

File : PC-BINK.EXE
Language : Turbo Pascal 5.5
Units : PCGEN
Extra files : PC-INK.DRV
Restrictions : HP DeskJet/ LaserJet II.

Instructions : Selecting invert prints white pixels as black and vice versa. Select frames on/off and resolution from menus.

Description : This program reads in a double size binary file and prints it on a HP printer. It has been found to work on the HP DeskJet and HP LaserJet IIP printers, but since it uses the HP printer language it should work on many of the HP printers.

File : PC-BPRIN.EXE
Language : Turbo Pascal 5.5
Units : PCGEN
Extra files : PC-PRINT.DRV
Restrictions : Dot matrix printers.

Instructions : Select the print orientation from the menu. Run PC-SETUP to change printer driver.

Description : This program reads in a double size binary file and prints it on a dot matrix printer. The current printer driver is used. The image can be printed vertically or horizantally and can be split over two pages for larger print outs.

File : PC-BSCRO.EXE
Language : Turbo Pascal 5.5
Units : PCGEN, GRAPH
Extra files : HERC.BGI
Restrictions : Only Hercules adaptor.

Instructions : When the image is displayed use the cursor keys to scroll the image up, down, left or right. Type 'Q' then enter to exit.

Description : This program reads in a double size binary image. The image is left in bit form, but in each byte the order of the bits is swapped for display purposes. The Turbo graphics routines are used to set the display apadter into graphics mode and therefore requires HERC.BGI. The image is then written directly into the display memory of a Hercules adapter.

File : PC-CONV.EXE
Language : Turbo Pascal 5.5
Units : PCGEN
Extra files : None
Restrictions : None

Instructions : Select type of conversion and file names.

Description : Converts between the various file formats (See programmer's section for description of each).

File : PC-EDGE.EXE
Language : Turbo Pascal 5.5
Units : PCGEN
Extra files : None
Restrictions : None

Instructions : Select CFI file and binary save file name. Then enter two threshold values. T2 > T1.

Description : A valley operator is used to find egdes. The likelihood of an edge is calculated on a 3x3 grid and compared to T1. If it is greater then the magnitude of the slope is calculated in eight directions. If any of these are greter than T2 then an edge point exists.

File : PC-FFT.EXE
Language : Turbo C 2.0

Include : PCGEN.H,RADIXT.H
Extra files : None
Restrictions : 1 MB Expanded memory required

Instructions : Select CFI file and save file name.

Description : A two dimensional FFT is performed on the input image and the resulting magnitude stored in the save file. The FFT is calculated using only integer values (for speed and storage space considerations). Due to the large amount of intermediate storage required, this program will only run on a compter with at least 1 Megabyte of available expanded memory (EMS version 3.2 or higher). On a 386 computer this program takes approximately 3 minutes to run (See PC-FFTT).

File : PC-FFTT.EXE
Language : Turbo C 2.0
Include : PCGEN.H,TMS320.H
Extra files : FFT512.B25
Restrictions : 1Mb Expanded memory, TMS320C20/25 card

Instructions : Select CFI file and save file name.

Description : A two dimensional FFT is performed on the input image and the resulting magnitude stored in the save file. The FFT is calculated using only integer values (for speed and storage space considerations). Due to the large amount of intermediate storage required, this program will only run on a compter with at least 1 Megabyte of available expanded memory (EMS version 3.2 or higher). This program uses the Delanco Spry TMS320C25 card to perform the FFTs (The base I/O should be set to 3E0 hex, and the memory address to D000 hex). On a 386 computer this program takes approximately 50 seconds to run.

File : PC-FILT3.EXE
Language : Turbo Pascal 5.5
Units : PCGEN,LISTFILE
Extra files : *.MSK, CONVOLVE.ASM, CONVOLVE.OBJ

Restrictions : None

Instructions : Select CFI file, save file name and filter mask.

Description : A 3x3 mask is convolved with the image. The mask is specified in a file which can be created with a text editor. Each element must be on a new line (i.e. nine lines in all). Assembler routines are used to do the convolution so this program executes faster than PC-FILTER. At this stage it is not guaranteed to be bug free.

File : PC-FILTE.EXE
Language : Turbo Pascal 5.5
Units : PCGEN,LISTFILE
Extra files : *.MSK
Restrictions : None

Instructions : Select CFI file, save file name and filter mask.

Description : A 3x3 mask is convolved with the image. The mask is specified in a file which can be created with a text editor. Each element must be on a new line (i.e. nine lines in all).

File : PC-GTOBI.EXE
Language : Turbo Pascal 5.5
Units : PCGEN
Extra files : None
Restrictions : None

Instructions : Select CFI file and a binary save file name.

Description : This program converts an input grey level file to a double size binary file by dithering with the Floyd-Steinberg algorithm [1]. The input image is increased to 1024 by 1024 pixels before conversion. The algorithm has been adapted to use integer values and so it will convert a .CFI file in approximately 1 minute on an AT. No speed improvement will result from using a numeric coprocessor.

References :

[1] Ulichney, Robert A., "Dithering with Blue Noise", *Proceedings of the IEEE*, Vol 76, No 1, January 1988.

File : PC-GTOI.EXE
Language : Turbo Pascal 5.5
Units : PCGEN
Extra files : None
Restrictions : None

Instructions : Select CFI file and a binary save file name.

Description : This program converts an input grey level file to a binary file by dithering with the Floyd-Steinberg algorithm [1]. The algorithm has been adapted to use integer values and so will convert a .CFI file in approximately 15 seconds on an AT. No speed improvement will result from using a numeric coprocessor.

References :

[1] Ulichney, Robert A., "Dithering with Blue Noise", *Proceedings of the IEEE*, Vol 76, No 1, January 1988.

File : PC-HIST.EXE
Language : Turbo Pascal 5.5
Units : PCGEN
Extra files : *.BGI
Restrictions : None

Instructions : Select CFI file.

Description : An assembler routine in PCGEN is used to count the occurence of each pixel value in the file. The histogram is scaled to fit on the screen and displayed. Black is on the left (0) and white on the right (255).

File : PC-HISTE.EXE
Language : Turbo Pascal 5.5
Units : PCGEN
Extra files : None
Restrictions : None

Instructions : Select CFI file and a save file name.

Description : This program equalizes the histogram of the input image. An assembler routine in PCGEN is used to calculate the histogram of the input file. It's cdf is calculated and matched with that of a flat histogram (Prof de Jagers M.Sc. Course notes) to generate a look up table. This is used to modify the image.

References :

[1] Gerhard de Jager, Multi Dimensional signal processing notes, M.Sc course, Electrical and Electronic Eng. Dept. UCT, 1989.

File : PC-IMAGE.EXE
Units : PCGEN, GRAPH
Extra files : *.BGI

This program reads in a binary file to a memory array using the routine in PCGEN. It then uses the Turbo graphics routines to draw the image on a Hercules, CGA, EGA/VGA adapter. The relevant .BGI file must be in the current directory.

File : PC-INK.EXE
Language : Turbo Pascal 5.5
Units : PCGEN
Extra files : PC-INK.DRV
Restrictions : HP DeskJet/ LaserJet II.

Instructions : Selecting invert prints white pixels as black and vice versa. Select frames on/off and resolution from menus.

Description : This program reads in a binary file and prints it on a HP printer. It has been found to work on the HP DeskJet and HP LaserJet IIP printers, but since it uses the HP graphics language it should work on many of the HP printers.

File : PC-LIST.EXE
Language : Turbo Pascal 5.5
Units : PCGEN, LISTFILE
Extra files : None
Restrictions : None

Instructions : Select directory.

Description : Displays list of all .CFI, .BIN, .PFI, .TXT files in specified directory.

File : PC-PRINT.EXE
Language : Turbo Pascal 5.5
Units : PCGEN
Extra files : PC-PRINT.DRV
Restrictions : Dot matrix printers.

Instructions : Select the print orientation from the menu. Run PC-SETUP to change printer driver.

Description : This program reads in a binary file and prints it on a dot matrix printer. The current printer driver is used. The image can be printed vertically or horizantally and can be split over two pages for larger print outs. The horizantal mode gives an aspect ratio close to that of the original television screen.

File : PC-SCROL.EXE
Language : Turbo Pascal 5.5
Units : PCGEN, GRAPH
Extra files : HERC.BGI
Restrictions : Only Hercules adaptor.

Instructions : When the image is displayed use the cursor keys to scroll the image up or down. Type 'Q' then enter to exit.

Description : This program reads in a binary image. The image is left in bit form, but in each byte the order of the bits is swapped for display purposes. The Turbo graphics routines are used to set the display adapter into graphics mode and therefore requires HERC.BGI. The image is then written directly into the display memory.

File : PC-SETUP.EXE
Language : Turbo Pascal 5.5
Units : PCGEN, LISTFILE
Extra files : *.DRV
Restrictions : None

Instructions : Select a printer from the menu.

Description : This program copies a selected printer.DRV file to PC-PRINT.DRV. Run this program before using PC-PRINT if you want to use a new printer. You can create your own printer drivers with a text editor. An example of the format is as follows:

```
IBM Graphics Printer
27/"*"/6/0/2
27/"*"/6/2/2
27/"*"/5/0/1
27/"*"/5/1/1
27/"@"
27/"A"/8/27/"2"
27/"A"/12/27/"2"
```

The first line is the printer name, the second the code to send 512 bytes of graphics to printer. The third is for 514 bytes (image + frame). Line 4 is for vertical graphics (512), line 5 (514 bytes), line 6 is for reset, line 7 for 1/8 line spacing, line 8 for 1/12 line spacing.

File : PC-SMOOT.EXE
Language : Turbo Pascal 5.5
Units : PCGEN
Extra files : None
Restrictions : None

Instructions : Select CFI file and save file name.

Description : A 3x3 mask is convolved with the image by an assembler routine. The mask is as below:

```
    1 1 1
1/9 * 1 1 1
    1 1 1
```

File : PC-SOBEL.EXE
Language : Turbo Pascal 5.5
Units : PCGEN
Extra files : None
Restrictions : None

Instructions : Select CFI file and save file name.

Description : Two 3x3 Sobel masks are convolved with the image by an assembler routine.

File : PC-STRET.EXE
Language : Turbo Pascal 5.5
Units : PCGEN
Extra files : None
Restrictions : None

Instructions : Select CFI file and save file name.

Description : The dynamic range of the image is calculated which is used to find the scale factor needed to generate a lookuptable. The lookuptable maps the old values of the pixels to new values such that the new image has the maximum dynamic range.

File : PC-STRET.EXE
Language : Turbo Pascal 5.5
Units : PCGEN
Extra files : None
Restrictions : None

Instructions : Select CFI file and save file name.

Description : The dynamic range of the image is calculated which is used to find the scale factor needed to generate a lookuptable. The lookuptable maps the old values of the pixels to new values such that the new image has the maximum dynamic range.

File : PC-THRES.EXE
Language : Turbo Pascal 5.5
Units : PCGEN
Extra files : None
Restrictions : None

Instructions : Select CFI file, BIN file and threshold.

Description : All pixels in the .CFI file whose value is greater or equal to the threshold are set to 1. all other pixels are set to 0. The method used by the program is create a 256 byte long lookup table with 0s up to the threshold value, and 1s thereafter. This is used to lookup a new value for each old pixel. This method is not the simplest, but it is the fastest because there are assembler routines in PCGEN for using such a lookup table.

File : PC-VIEW2.EXE
Language : Turbo Pascal 5.5

Units : PCGEN
Extra files : None
Restrictions : Only Monochrome VGA.

Instructions : Type Esc to exit.

Description : This program reads in a gray level file. The image is displayed using 16 levels on a monochrome VGA adapter using the standard 640x480x16 VGA colour mode.

File : PC-VIEW5.EXE
Language : Turbo Pascal 5.5
Units : PCGEN
Extra files : None
Restrictions : Tseng-Lab VGA adapter.

Instructions : Type Esc to exit.

Description : This program reads in a gray level file. The image is displayed using 16 or 256 levels on a Tseng-lab (or compatible) VGA using non-standard video modes up to 800 by 600 by 256 levels. The BIOS routines for setting the video mode and displaying pixels are called directly so there is no need for the Turbo graphics (they cannot handle these modes anyway). The values in each of the colour registers are set to gray levels, so even on a colour display the image appears in monochrome. In the 800 by 600 mode a histogram is displayed next to the image. See Ultra-VGA adapter reference booklet for further details of video modes.

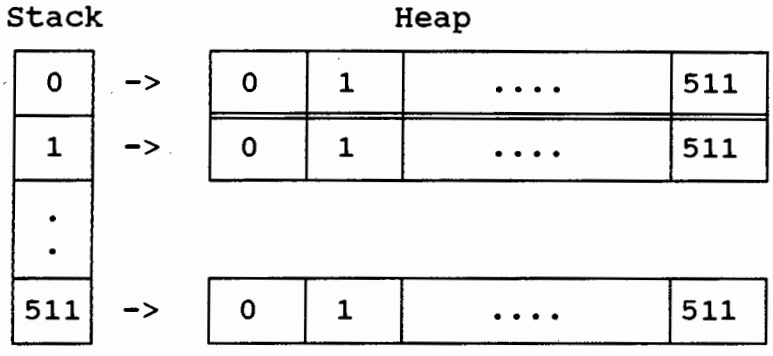
3. Programmer's Guide

3.1 Data Structures

In Turbo Pascal the data area (Data Segment) where variables are kept is only 64kb. In practical terms this would limit the size of an image array to a maximum of around

100x100 pixels. To overcome this limitation the heap is used to store the image (for more information on the heap see the Turbo manual).

An array of 512 pointers is kept in the data area (requiring only 2Kb of the data segment). Each pointer points to a 512 array of bytes on the heap (requiring a total of 256Kb on the heap).



Example:

The pascal statement to read a value from a picture m, at row 100 and column 25 would be:

```
pixel:=m[100]^[25];
```

3.2 File Formats

There are four different file formats supported. Two of these are required for interfacing to other programs. The other two are internal. The two internal formats are picture files and binary files. These files are given the extension .PFI and .BIN respectively. The other formats are camera files (.CFI) and ASCII files (.TXT). The .CFI files can be directly displayed on many frame grabbers while the .TXT files can be imported into spreadsheets of word processors. Detailed descriptions of each file format follows:

.CFI files

These files are used to store 8 bit gray level images with 512 rows by 512 columns. The file is simply a stream of 262144 bytes with no formatting characters.

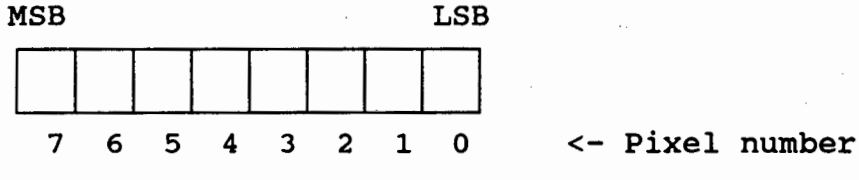
.PFI files

These files are used to store gray level images (8 bits) of any specified number of rows and columns up to a maximum of 512x512. At the start of the file is a header of 512 bytes. The first two bytes represent an unsigned integer which gives the number of rows in the file. The next two bytes give the number of columns. The following byte gives the number of bits/pixel. For 8 bit images this should be set to eight. The remainder of the header is reserved at this stage and should be set to zeros when writing the file. Following the header is the stream of byte values (with no formatting characters).

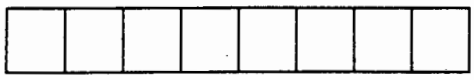
.BIN files

These files are used to store 512 x 512 x 1 bit images. Therefore they can only be used to store black and white images (such as would result from thresholding a .CFI file). The file consists of 32768 bytes. 64 bytes represents each row, with 512 such rows. Each byte contains 8 bits representing 8 pixels. However, the order of the bits is reversed. For example, if an image is labelled with (0,0) in the top left corner, pixel (0,0) would be the LSB of the first byte in file. This is to remain compatible with the VAX Pascal 'pack' and 'unpack' commands.

In a byte:



On the screen:



.TXT files

The gray level values are stored as ASCII digits e.g. 100 is stored as '1','0','0'. Each pixel value is separated by a space. e.g. 255 254 30 1 128 etc... The end of a row is denoted by a return (EOLN) and the end of the file by an EOF marker. There is no header information for this type of file. These files may be imported into Lotus or AsEasyAs for analysis.

3.3 Available Functions and Procedures

The unit PCGEN.PAS contains the routines that can be called from any Turbo Pascal program. A version of this file is being written in Turbo C but not all functions are available yet.

The following is a print out of the interface section of the PCGEN unit (Turbo Pascal 5.5):

```
uses dos,crt,graph,menu,listfile,entry;

type mrow=array[0..511] of byte;
     mptr=^mrow;
     matrix=array[0..511] of mptr;

    buffertype=array[0..127] of byte;
    bptr=^buffertype;
    bintype=array[0..1023] of bptr;

const fonts: array[0..4] of string[13] =
('Defaultfont','Triplexfont',
```

```
'Smallfont','Sansseriffont',
                                              'Gothicfont');
      Linestyles: array[0..4] of string[9] =
('Solidln','dottedln',

'centreln','dashedln','Userln');
      Textdirect:array[0..1] of string[8] = ('Hdir','Vdir');

var graphmode,graphdriver:integer;
    maxcolour:word;
    errcode:integer;
    maxx,maxy:integer;

procedure initgraphics;
{ Initialise Turbo Graphics }

procedure openstatus(f,p:string;mrows:word);
procedure status(row:integer);
procedure closestatus;

procedure solidline(y:integer);
{Draw a solid line across the screen}

procedure parse(var name:string;defext:string);
{Parse a filename and add default extension if necessary}

procedure filenames(var f1,f2:string;var escape:boolean);
{ get two file names - must pass mask in f1, ext in f2 }

procedure title(s:string);

procedure setboolean(var b:boolean;s:string;var
escape:boolean);

procedure sl(s:string;wait:boolean);
```

```
procedure lookup(b,l:pointer;bytes:word);
procedure pack(m,b:pointer;bytes:word);
procedure unpack(b,m:pointer;bytes:word);

procedure declarepicture(var m:matrix);
procedure savecfi(filename:string;m:matrix);
procedure loadcfi(filename:string;var m:matrix);
procedure savepfi(filename:string;m:matrix;rows,cols:word);
procedure loadpfi(filename:string;var m:matrix;var
rows,cols:word);
procedure loadbin(filename:string;var m:matrix);
procedure savebin(filename:string;m:matrix);
procedure loadbbin(filename:string;var binbuff:bintype);
procedure savebbin(filename:string;binbuff:bintype);

procedure findrange(m:matrix;rows,cols:word;var max,min:byte);
procedure stretchrange(m:matrix;rows,cols:word);
```

Units

The units required for compliation are : MENU.TPU, LISTFILE.TPU, FINDFILE.TPU and the Turbo graphics unit GRAPH.TPU.

Description of procedures

Initgraphics: Loads the appropriate graphics driver and sets the adapter into graphics mode. Sets MaxX, MaxY and MaxColour.

Parameters: None

Example: Initgraphics;

OpenStatus: Initialise the status window in the centre of the screen.

Parameters: 1. Current File name (string)
2. Current process (string)

3. Total number of rows (word)

Example: OpenStatus(Filename,'Convolve',512);

Status: Update the status window in the centre of the screen. OpenStatus must be called first.

Parameters: 1. Current Row (word)

Example: Status(Row);

CloseStatus: Close the status window in the centre of the screen. OpenStatus must be called first.

Parameters: None

Example: CloseStatus;

SolidLine: Draw a solid line across the screen at the row specified:

Parameters: 1. Row (integer)

Example: SolidLine(2);

Parse: Checks if the filename already has an extension. If not the default extension is added.

Parameters: 1. File name (Var string)

2. Default extension (string)

Example: Parse(filename,'CFI');

Filenames: Get one or two file names from the user. If parameters were passed on the command line these will be returned. The first parameter specifies the mask on entry and returns the first filename. The second parameter must be set to '' if no other file is required. If a second file is required then parameter two specifies the default extension for this second name on entry and contains the full name on exit. If the user presses Esc during the procedure the variable 'escape' will be set to true.

Parameters: 1. File name 1 (Var string)

2. Default extension for file name 2 (Var string)

3. Escape pressed (Var boolean)

Example: Filenames(filename1,filename2,escape);

Title: Clears the screen then displays the title at the top of the screen with a solid line below it. It also sets the display colours needed for the rest of the program and so it should always be called at least once.

Parameters: 1. Title (string)

Example: Title('PC-FILTER');

SetBoolean: Displays a menu with the options Yes and No. The boolean variable is set to true or false depending on the choice. If the user presses Esc, the variable 'escape' will be set to true.

Parameters: 1. Variable to be set (Var boolean)
2. Menu heading (string)
3. Escape (Var boolean)

Example: SetBoolean(invert,'Invert image',escape);

Sl: Print a status line at the bottom of the screen. If wait is true then the user must press a key to continue.

Parameters: 1. Message (string)
2. Wait (boolean)

Example: Sl('Convolving with Sobel Masks',false);

LookUp: Replaces bytes in a data array by using a lookup table to find a new value. The routine is written in assembler and is extremely compact and fast.

Parameters: 1. Data (pointer)
2. Lookup table (pointer)
3. Bytes to process (word)

Example: LookUp(m[row],@lookuptable,512);

Pack: Packs eight bytes (each 1 or 0) into 8 bits, and reverses the order for compatibility with the VAX Pascal PACK command. Uses assembler routines for speed.

Parameters: 1. Byte Data (pointer)
2. Bit Data (pointer)
3. Bytes to process/8 (word)

Example: Pack(m[row],@binary[row],64);

UnPack: Unpacks eight bits into 8 bytes, and reverses the order for compatibility with the VAX Pascal UNPACK command. Uses assembler routines for speed.

Parameters: 1. Bit Data (pointer)
2. Byte Data (pointer)
3. Bytes to process/8 (word)

Example: UnPack(@binary[row],m[row],64);

DeclarePicture: Reserves 512 rows of 512 columns on heap and initialises all elements to zero.

Parameters: 1. Image (matrix)

Example: DeclarePicture(m);

SaveCFI: Saves image in memory to disk in the .CFI file format. The image in memory is not affected.

Parameters: 1. Filename (string)
2. Image (matrix)

Example: SaveCFI(filename,m);

LoadCFI: Loads .CFI image from disk to memory. The routine declares it's own memory space for the image.

Parameters: 1. Filename (string)
2. Image (Var matrix)

Example: LoadCFI(filename,m);

SavePFI: Saves image in memory to disk in the .PFI file format. The image in memory is not affected.

Parameters: 1. Filename (string)
2. Image (matrix)
3,4: Rows, Columns (2x word)

Example: SavePFI(filename,m,rows,cols);

LoadPFI: Loads .PFI image from disk to memory. The routine declares it's own memory space for the image.

Parameters: 1. Filename (string)
2. Image (Var matrix)
3,4: Rows, Columns (2x Var word)

Example: LoadPFI(filename,m,rows,cols);

SaveBIN: Saves image in memory to disk in the .BIN file format. The image in memory is not affected.

Parameters: 1. Filename (string)
2. Image (Var matrix)

Example: SaveBIN(filename,m);

LoadBIN: Loads .BIN image from disk to memory. The routine declares it's own memory space for the image.

Parameters: 1. Filename (string)
2. Image (matrix)

Example: LoadBIN(filename,m);

SaveBBIN: Saves image in memory to disk in the .DBN file format. The image in memory is not affected.

Parameters: 1. Filename (string)
2. Image (Var matrix)

Example: SaveBBIN(filename,m);

LoadBBIN: Loads .DBN image from disk to memory. The routine declares it's own memory space for the image.

Parameters: 1. Filename (string)
2. Image (matrix)

Example: LoadBBIN(filename,m);

FindRange: Find the dynamic range of the image. Returns the maximum and minimum gray levels in the image.

Parameters: 1: Image (matrix)
2,3: Rows,Columns (2x word)
4,5: Max,Min (2x byte)

Example: Findrange(m,512,512,max,min);

StretchRange: Stretch the dynamic range of the image. Calls FindRange to find maximum and minimum gray levels in the image, then generates lookuptable which is then used to modify the image.

Parameters: 1: Image (matrix)

2,3: Rows,Columns (2x word)

Example: StretchRange(m,512,512);

3.4 Adding programs to the menu

When the user types PC-MENU, a batch file is run which displays a list of programs. The list contains all the programs in the current directory that match the file spec 'PC-*.EXE'. So to make a program appear in the list make sure that it's name matches. (If your program is a .COM file, just rename it as .EXE)

APPENDIX B

Table of Contents

1. Listing of HOUGH.C

```
/*************************************************************/
/* Hough Transform                                           */
/* Written in Turbo C version 2.0                            */
/* On the 11th July 1990                                     */
/*                                                           */
/* This program implements the straight line version of the */
/* Hough Transform (Duda and Hart, 1972)                     */
/*                                                           */
/* Extra Hardware:                                           */
/*   640k memory, at least 590k free                         */
/*************************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#include "pcgen.h"              /* General purpose functions  */

unsigned char
     *m[512],                   /* Array to hold image        */
     *A[512];                   /* Accumulator array          */
int  error;                     /* Error code            */

main()
{
  char filename[40],                /* Input filename         */
       filename2[40];               /* Output filename        */

  printf("Enter load file name (.BIN) :");
  scanf("%s",filename);
  printf("Enter save file name (.CFI) :");
  scanf("%s",filename2);

  loadbin(filename,m);   /* Load binary file (in pcgen.h)*/
  error = hough();       /* Hough transform       */
  if (error==FALSE)
    savecfi(filename2,A);/* Save .CFI file    ( "    "  )*/
  else
    printf("Memory full error\007\n");
}


int hough()
```

```
/*************************************************************/
/*  1. Clear accumulator                                     */
/*  2. Initialise sine and cosine tables                     */
/*  3. For all rows                                          */
/*    3.1 For all columns                                    */
/*        3.1.1 If image(row,column) not equal 0             */
/*           For all angles                                  */
/*             p = x * cos(th)+ y * sin(th)                  */
/*             increment (A[p,th])                           */
/*************************************************************/
{
  int rho,th,row,col;
  float theta;
  int x,y;
  unsigned char *tp;
  int memerror;
  float a;
  int sine[512],cosine[512];  /* Sine and cosine tables     */

  memerror = FALSE;

  printf("Initialise Hough space\n");

  for (rho=-256;rho<256;rho++){
    if ((A[rho+256] = (unsigned char *)
                          calloc(512,sizeof(char)))==NULL){
      memerror=TRUE;}    /* Allocate memory and set to 0 */
  }

  a = PI/512;             /* Initialise sin and cos tables */
  for (th=0;th<512;th++){
    theta = th * a;
    cosine[th] = cos(theta) * 128.0 ;
    sine[th]   = sin(theta) * 128.0 ;
  }

  if (memerror==FALSE){
    printf("Hough Transform\n");
    printf("Row:");
    for (y=-255;y<=256;y++){
      if (((y+256)%8)==7) printf("%4d\b\b\b\b",y+257);
      tp = m[256-y];          /* temporary pointer to row */
      for (x=-256;x<256;x++){
     if (*tp!=0) {
       for (th=0;th<512;th++){
         rho = (x * cosine[th] + y * sine[th])/ 128 ;
         if ((rho>-256) && (rho<256))
           *(A[256-rho]+th)+=1;    /* Increment accumulator */
       }
     }
     tp++;                  /* point to next image element */
      }
    }
    printf("%4d\n",512);
```

```
    }
    return(memerror);
}
```

2. Listing of RADON.C

```
/*************************************************************/
/* Radon Transform                                           */
/* ===============                                           */
/*                                                           */
/* Written in Turbo C version 2.0 uses EMS                   */
/*   Optimised for speed, register variables and 80286 code  */
/* Extra Hardware:                                           */
/*      1 Mbyte of Expanded Memory Ver. 4.0 or higher        */
/*                                                           */
/*************************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <dos.h>

#include "ems.h"          /* Expanded memory functions  */
#include "pcgen.h"        /* General purpose functions  */

#define PICSIZE 512       /* Image size                 */
#define PI     3.1415926535   /* Numerical constant         */
#define   FALSE     0         /* Logical constants          */
#define TRUE   !FALSE
#define EMSTYPE     long      /* EMS pixel type (32 bits)   */

unsigned char  *m[512];  /* Image array (8 bits / pixel) */
long handle,             /* EMS handle                   */
     stime,etime;        /* Start and end times          */
char errmsg[12][12];     /* Error messages               */
int  currentpage,        /* Current EMS page in page frame*/
     rpp,                /* Rows per EMS page            */
     pages;              /* Total number of EMS pages used*/


main()
{
  int     error;              /* Error code              */
  char    filename[40],       /* Input file name         */
          filename2[40];      /* Output file name        */

  strcpy(errmsg[0],"Completed");
  strcpy(errmsg[1],"File error");
  strcpy(errmsg[2],"Mem error");

  rpp = 16 * 2 / sizeof(EMSTYPE);     /* Rows per EMS page */
  pages = 512 / 2 * sizeof(EMSTYPE) / 16;  /* pages needed */
  if (emmchk())               /* EMS manager present?  */
    if (emmok())              /* EMS working ok?  */
      if ((handle = emmalloc(pages))>=0){ /* Allocate pages */
```

```
      printf("Radon Transform\n");
      printf("Enter load file name (.CFI) :");
      scanf("%s",filename);
      printf("Enter save file name (.CFI) :");
      scanf("%s",filename2);

      stime = time(NULL);

      emmmap(handle,0,0); /* Initialise to page 0 */
      currentpage = 0;

      error = loadcfi(filename,m);        /* Load image       */
      if (error==0){
        radon();                          /* Radon transform  */
        savecfi(filename2,m);             /* Save result      */
        }
      else printf("Error Loading : %s",errmsg[error]);

      emmclose(handle);
      etime = time(NULL)-stime;
      printf("Time: %05d seconds",etime);

      }
      else printf("Expanded memory error\n%4d pages
needed.\n",pages);
}


EMSTYPE *getrowadd(int row)
/*************************************************************/
/* Returns physical memory address of row                    */
/* Change this routine for different implementations of EMS */
/*************************************************************/
{
      int  page;
      char      *tp;

      if ((page = row / rpp)!=currentpage){
        emmmap(handle,0,page);
        currentpage = page;
        }
      tp = emmbase + (row%rpp)*512*sizeof(EMSTYPE);
      return(tp);
}
```

```
clearems()
/*****************/
{
      int  row,col;
      EMSTYPE   *ro;

      for (row=0;row<512;row++){
        ro = getrowadd(row);
        for (col=0;col<512;col++)
          *(ro+col) = 0;
        }
}


unloadems()
/*****************************************/
/* Take Magnitude and scale to fit into m */
/*****************************************/
{
      EMSTYPE   max,min,temp;
      int  row,col,real,imag;
      EMSTYPE   far *ro;
      float     scale;

      /* Scale results back to 8 bits */
      ro = getrowadd(0);
      max = *ro;
      min = max;
      printf("Find dynamic range :");
      for (row=0;row<512;row++){
        if ((row%8)==0)
          printf("%4d\b\b\b\b",row);
        ro = getrowadd(row);
        for (col=0;col<512;col++){
          temp = *(ro+col);
          if (temp>max) max = temp;
          if (temp<min) min = temp;
          }
        }
      printf("%4d (%+ld..%+ld)\n",512,min,max);

      if (max == min)
            scale = 0;
        else scale = 255.0/(max-min);
      printf("Scale factor : %e\n",scale);
      printf("Scale :");
      for (row=0;row<512;row++){
        if ((row%8)==0) printf("%4d\b\b\b\b",row);
        ro = getrowadd(row);
        for (col=0;col<512;col++){
          temp =*(ro+col)-min;
          if (temp==0)
            *(m[row]+col) = 0;
          else
```

```
            *(m[row]+col) = temp * scale;
          }
        }
      printf("%4d\n",512);
}


radon()
{
  int rho,theta,row,col;
  int x,y;
  float a;
  int sine[512],cosine[512];
  unsigned char *tp;
  EMSTYPE *ro;

  printf("Initialise\n");

  clearems();

  a = PI/512;
  for (theta=0;theta<512;theta++){
    cosine[theta] = cos(theta*a) * 128.0 ;
    sine[theta]   = sin(theta*a) * 128.0 ;
    }

  printf("Radon Transform\n");
  printf("Row:");
  for (y=-255;y<=256;y++){
    printf("%4d\b\b\b\b",y+255);
    tp = m[256-y];
    for (x=-256;x<256;x++){
      if (*tp!=0){
     for (theta=0;theta<512;theta++){
     ro =
 getrowadd((int)((unsigned int)(32767-x*cosine[theta]-
y*sine[theta]) >> 7));
       *(ro+theta)+=*tp;      /* Add gray level   */
       }
     }
      tp++;
      }
    }
  printf("%4d\n",512);
  unloadems();
}
```

3. Listing of PST.C

```
/*************************************************************/
/* Inverse Radon Transform using the projection slice        */
/* theorem.                                                  */
/* Written in Turbo C version 2.0 uses 1MB of EMS            */
/*   Optimised for speed, register variables and 80286 code  */
/* Extra Hardware:                                           */
/*   1 M byte of Expanded memory                             */
/*   Dalanco Spry TMS320C25 plug in card                     */
/*                                                           */
/*************************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <dos.h>

#include "ems.h"          /* Expanded memory manager functions
*/
#include "pcgen.h"        /* General purpose functions       */
#include "tms320.h"       /* TMS320C25 functions (FFT)       */

#define PICSIZE 512       /* Define Image size               */
#define PI     3.1415926535   /* Numerical constant
*/
#define   FALSE     0         /* Logical constant            */
#define TRUE   !FALSE

unsigned char  *m[512];   /* Image array (8 bits/pixel)      */
long handle,              /* EMS handle                      */
     stime,etime;         /* Start and finish times          */
char errmsg[12][12];      /* Error messages                  */
int  currentpage;         /* EMS page currently in frame     */



main()
{
  int error;                    /* Error code                */
  char filename[40],            /* Input file name           */
       filename2[40];           /* Output file name          */

  strcpy(errmsg[0],"Completed");
  strcpy(errmsg[1],"File error");
  strcpy(errmsg[2],"Mem error");


  if (emmchk())                 /* Is EMS manager present?   */
    if (emmok())                /* Is it working ok?         */
      if ((handle = emmalloc(64))>=0){  /* Allocate pages    */
```

```
      printf("Inverse Radon Transform\n");
      printf("Enter load file name (.CFI) :");
      scanf("%s",filename);
      printf("Enter save file name (.CFI) :");
      scanf("%s",filename2);

      stime = time(NULL);

      emmmap(handle,0,0);       /* Initialise to page 0        */
      currentpage = 0;

      if ((loadcfi(filename,m))==0){     /* Load image file  */
        irad();              /* Inverse transform           */
        savecfi90(filename2,m); /* Save file (rotate by 90)  */
        }
      emmclose(handle);
      etime = time(NULL)-stime;
      printf("Time: %05d seconds",etime);

      }
      else printf("Expanded memory error\n");
}


int *getrowadd(int row)
/*************************************************************/
/* Returns physical memory address of row                    */
/* Change this routine for different implementations of EMS */
/*************************************************************/
{
      int  page;
      char      *tp;

      if ((page = row / 8)!=currentpage){
        emmmap(handle,0,page);
        currentpage = page;
        }
      tp = emmbase + (row%8)*2048;
      return(tp);
}
```

```
irad()
/*****************************************************************/
/*  1. First transpose input (m) and put it into EMS             */
/*  2. 1D FFT of each row                                        */
/*  3. Interpolate onto cartesian coordinates                    */
/*     results saved to disk )                                   */
/*     Load result, discard 1D FFTs                              */
/*  4. 2D IFFT of above                                          */
/*  5. Scaling back into m.                                      */
/*****************************************************************/

{


     sincostables();
     /* Step 1. Load EMS and transpose */
     tloadems(m);

     /* Step 2. Take row transforms of image and put results
in EMS */
     onedrowffts();
     shifttocentre();

     /* Step 3. Interpolate onto square grid */
     interpolate();

     /* Step 4. 2D IFFT */


     onedrowiffts();
     transpose();
     onedrowiffts();
     twodshift();

     /* Step 5. Scale down results */
     unloadems(m);

}


tloadems(unsigned char *m[512])
/*******************************/
/* Load m into EMS,transpose   */
/*******************************/
{
     int  row,col;
     int  *ro;

     for (row=0;row<512;row++){
       ro = getrowadd(row);
       for (col=0;col<256;col++){

         *(ro + col + 256) = *(m[col] + row) * 128;
/* Transpose real */
```

```
          *(ro + col) = *(m[col+256] + row) * 128;
/* Transpose real */
          *(ro + 512 + col) = 0;          /* Imaginary */
          *(ro + 512 + col+256) = 0;      /* Imaginary */
            }
        }
}


unloadems(unsigned char *m[512])
/*****************************************/
/* Take Magnitude and scale to fit into m */
/*****************************************/
{
     int  max,min,temp,row,col,real,imag;
     int  far *ro;
     int  scale;

     /* Scale results back to 8 bits */
     max = 0;
     min = 0;
     printf("Find dynamic range :");
     for (row=0;row<512;row++){
       if ((row%8)==0)
         printf("%4d\b\b\b\b",row);
       ro = getrowadd(row);
       for (col=0;col<512;col++){
         real = *(ro+col);
         imag = *(ro+512+col);


         if (real == 0)
           temp = abs(imag);
         else
           if (imag == 0)
          temp = abs(real);
           else
          temp = sqrt((long)real * real + (long)imag * imag +1
);

         if ((abs(row-256)<3))
           if (temp>max) temp = max;

         if (temp>max) max = temp;
         if (temp<min) min = temp;
         *(ro+col) = temp;
         }
       }
     printf("%4d (%+d..%+d)\n",512,min,max);

     if (max == min)
           scale = 0;
       else scale = (255*128)/(max-min);
     printf("Scale factor : %5.2f\n",scale/128);
```

```
      printf("Scale :");
      for (row=0;row<512;row++){
        if ((row%8)==0) printf("%4d\b\b\b\b",row);
        ro = getrowadd(row);
        for (col=0;col<512;col++){
          temp =*(ro+col)-min;
          if (temp==0)
            *(m[row]+col) = 0;
          else
            *(m[row]+col) = temp * scale / 128;
          }
        }
      printf("%4d\n",512);
}


shifttocentre()
/*************************************************************/
/* One dimensional shift to centre                           */
/* Shift each column to centre so that D.C is in the centre */
/* of the image                                              */
/*************************************************************/
{
      int row,col;
      int  *ro,*bf;

      bf = (int *)calloc(1024,sizeof(int));

      printf("Shift to centre");
      for (row=0;row<512;row++){
        ro = getrowadd(row);
        memcpy(bf,ro,1024*sizeof(int));
        for (col=0;col<256;col++){
          *(ro+col+256) = *(bf+col);
          *(ro+col) = *(bf+col+256);
          *(ro+col+256+512) = *(bf+col+512);
          *(ro+col+512) = *(bf+col+256+512);
          }
        }
        free(bf);
        printf("\n");
}


twodshift()
/*************************************************************/
/* Two dimensional shift to centre. D.C in the centre of the*/
/* image                                                     */
/*************************************************************/
{
      int row,col;
      int  *ro,*bf1,*bf2;

      printf("2D Shift to Centre");
```

```
      bf1 = (int *)calloc(1024,sizeof(int));
      bf2 = (int *)calloc(1024,sizeof(int));

      for (row=0;row<256;row++){

        /* first swap top right with bottom left */
        ro = getrowadd(row+256);
        memcpy(bf1,ro,1024*sizeof(int)); /* save bottom */
        ro = getrowadd(row);
        memcpy(bf2,ro,1024*sizeof(int)); /* save top */

        memcpy(ro+256,bf1,256*sizeof(int));
                /* top right = bottom left */
        memcpy(ro+256+512,bf1+512,256*sizeof(int));

        memcpy(ro,bf1+256,256*sizeof(int));
                /* top left = bottom right */
        memcpy(ro+512,bf1+256+512,256*sizeof(int));

        ro=getrowadd(row+256);
        memcpy(ro,bf2+256,256*sizeof(int));
                /* bottom left = top right */
        memcpy(ro+512,bf2+256+512,256*sizeof(int));

        memcpy(ro+256,bf2,256*sizeof(int));
                /* bottom right = top left */
        memcpy(ro+256+512,bf2+512,256*sizeof(int));
        }
        free(bf1); free(bf2);
        printf("\n");
}


interpolate()
/*************************************************************/
/* Zero order interpolation                                  */
/*************************************************************/
{
#define SINTYPE int

      int  x,y,rho,th;
      int  far *bf1,*bf2,*ro;
      FILE *fp;
      float     a;
      SINTYPE   *sine,*cosine;

      printf("Interpolate\nRow :");


      sine   = (SINTYPE *)calloc(512,sizeof(SINTYPE));
      cosine = (SINTYPE *)calloc(512,sizeof(SINTYPE));

      a = PI / 512.0;
      for (th=0;th<512;th++){
```

```
    *(sine+th)   =  sin(th*a) * 128;
    *(cosine+th) =  cos(th*a) * 128;
    }

  bf1 = (int *)calloc(1024,sizeof(int));
  bf2 = (int *)calloc(1024,sizeof(int));
  fp = tmpfile();


  for (y=0;y<256;y++){
    if ((y%4)==0) printf("%4d\b\b\b\b",y*2);
    memset(bf1,1024,0);
    memset(bf2,1024,0);

    for (x=0;x<256;x++){
      th = (x!=0)?atan2(y,x)/a:256;
      if (th>512) th-=512;
      if (th<0) th+=512;
      rho = (x * *(cosine+th) + y * *(sine+th)) / 128;
      ro = getrowadd(th);
      *(bf1 + x + 256      ) = *(ro + rho + 256);
      *(bf1 + x + 256 + 512) = *(ro + rho + 256 + 512);
      *(bf2 - x + 256      ) = *(ro - rho + 256);
      *(bf2 - x + 256 + 512) = *(ro - rho + 256 + 512);
      ro = getrowadd(512-th);
      *(bf1 - x + 256      ) = *(ro + rho + 256);
      *(bf1 - x + 256 + 512) = *(ro + rho + 256 + 512);
      *(bf2 + x + 256      ) = *(ro - rho + 256);
      *(bf2 + x + 256 + 512) = *(ro - rho + 256 + 512);
    }
    fwrite(bf1,sizeof(int),1024,fp);
    fwrite(bf2,sizeof(int),1024,fp);
  }
   printf("%4d\nReading data back\n",512);
   rewind(fp);
   for (y=0;y<512;y++){
  ro = getrowadd(y);
  memset(ro,0,1024*sizeof(int));
  }
   for (y=0;y<256;y++){
  ro = getrowadd(y+256);
  fread(ro,sizeof(int),1024,fp);

  ro = getrowadd(-y+255);
  fread(ro,sizeof(int),1024,fp);
  }
   free(sine); free(cosine); free(bf1); free(bf2);
   fclose(fp);
   printf("Completed\n");


}
```

```
onedrowffts()
/******************/
{
      int  row,col,real,imag;
      int  far *ro;

      printf("Forward FFT :");
      for (row=0;row<512;row++){
        if ((row%8)==0)
          printf("%4d\b\b\b\b",row);

        ro = getrowadd(row);
        complexfft(ro,ro+512);  /* start FFT */
        }
      printf("%4d\n",512);
}



onedrowiffts()
/*********************************/
/* Parallel some of the processes */
/*********************************/
{
      int  max,temp,row,col,real,imag;
      int  far *ro;
      int  scale;

      max = 1;
      printf("Scale :");
      for (row=0;row<512;row++){
        ro =getrowadd(row);
        for (col=0;col<512;col++){
          real = abs(*(ro+col));
          imag = abs(*(ro+col+512));
          temp = (real>imag)?real:imag;
          max  = (temp>max)?temp:max;
          }
        }

      if (max<4096)
        scale = 2*4096.0 / max + 0.5;
        else scale = 2;

      printf("(%4d..%4d) %e\n",0,max,scale/2.0);

      printf("Inverse FFT :");

      ro=getrowadd(0);
      for (col=0;col<512; col++){
        *(ro+col) = ((long)*(ro+col) * scale /2);
        *(ro+col+512) = (*(ro+col+512) * scale /2);
```

```
      }
    startfft(ro,ro+512);

    for (row=1;row<512;row++){
     if ((row%8)==0)
       printf("%4d\b\b\b\b",row);

      ro = getrowadd(row);
      for (col=0;col<512; col++){
        *(ro+col) = *(ro+col) * scale;
        *(ro+col+512) = *(ro+col+512) * scale;
        }

      ro = getrowadd(row-1);
      stopfft(ro,ro+512);

      for (col=0; col<512; col++)
        *(ro+col+512)-=0;

      ro=getrowadd(row);
      startfft(ro,ro+512);  /* start FFT */

      }

    ro = getrowadd(511); /* find mag of last row */
    stopfft(ro,ro+512);

    printf("%4d\n",512);
}


transpose()
/*************************************************************/
/* Transpose Matrix in EMS, Handle given by handle           */
/* Eklundh's transposition algorithm                         */
/*************************************************************/
{
    unsigned int   a,b,c,d,x,y,row,col;
    int     *ro,*bf1r,*bf1i,*bf2r,*bf2i;
    int  r1,r2,i1,i2;
    unsigned int bs,mb,rb;


    bf1r = (int*)malloc(512*sizeof(int));
    bf1i = (int*)malloc(512*sizeof(int));
    bf2r = (int*)malloc(512*sizeof(int));
    bf2i = (int*)malloc(512*sizeof(int));

    printf("Transpose :");

/* Optimise for large blocks */
    for (bs=256;bs>1;bs>>=1){
      rb = 256/bs;
      printf("%4d\b\b\b\b",bs);
```

```
        for (x=0;x<rb;x++)
          for (y=0;y<rb;y++){
            c = x*2*bs;
            b = y*2*bs;
            a = c+bs;
            d = b+bs;
            mb = bs * sizeof(int);
            for (row=0;row<bs;row++){

           /* Fetch source data */
           ro = getrowadd(row+b) + a;
           memcpy(bf2r,ro,mb);       /* Real */
           memcpy(bf2i,ro+512,mb);   /* Imag */

           /* Save destination data */
           ro = getrowadd(row+d) + c;
           memcpy(bf1r,ro,mb);       /* Real */
           memcpy(bf1i,ro+512,mb);   /* Imag */

           /* Write to destination */
           /* ro = getrowadd(row+d) + c; */ /* Already got it*/
           memcpy(ro,bf2r,mb);       /* Real */
           memcpy(ro+512,bf2i,mb);   /* Imag */

           /* Write saved dest to source */
           ro = getrowadd(row+b) + a;
           memcpy(ro,bf1r,mb);       /* Real */
           memcpy(ro+512,bf1i,mb);   /* Imag */
           }
            }
        }

/* Optimize for single integer (i.e bs=1) */
        if (bs==1){
        printf("%4d\b\b\b\b",bs);
        for (x=0;x<256;x++)
          for (y=0;y<256;y++){
            c = x*2;
            b = y*2;
            a = c++;
            d = b++;

            /* Fetch source data */
            ro = getrowadd(b) + a;
            r2 = *ro;
            i2 = *(ro+512);

            /* Save destination data */
            ro = getrowadd(d) + c;
            r1 = *ro;         /* Real */
            i1 = *(ro+512);   /* Imag */

            /* Write to destination */
            /* ro = getrowadd(d) + c; */ /* Already got it! */
```

```
            *ro = r2;
            *(ro+512) = i2;

            /* Write saved dest to source */
            ro = getrowadd(b) + a;
            *ro = r1;
            *(ro+512) = i1;
            }
        }
      printf("\n");
      free(bf1r);free(bf1i);free(bf2r);free(bf2i);
}


savecfi90(char name[],unsigned char *m[512])
/* Routine to save contents of memory array as .CFI file-
rotate by 90 */
{
  int row,col,index;
  char *buffer;
  FILE *fp;

  printf("Writing %s\n",name);

  if ((buffer = (char*)malloc(4096))!=NULL){
    printf("Row:");
    fp = fopen(name,"wb");
    for (row=0;row<512;row++){
      printf("%4d\b\b\b\b",row);
      for (col=0;col<512;col++)
       *(buffer+512-col) = *(m[col]+511-row);
      fwrite(buffer,sizeof(char),512,fp);
      }
    fclose(fp);
    printf("%4d\n",512);
    free(buffer);
  }
  else printf("Memory allocation error\007\n");

}
```

4. Listing of IRADE.PAS

```
program iradonfft(input,output);
{ Inverse Radon transform using the projection slice theorem }
{ Zero order interpolation is used                           }
{ Runs on the Vax and uses IEEE Fortran FFT routines         }

const
     picsize=511;
     interim=true;

type byte=[byte] 0..255;
     bit=[bit] 0..1;
     string=varying[80] of char;
     buffertype=packed array[0..picsize] of byte;

     slice=array[0..513] of real;
     smslice=array[1..512] of real;
     fftarray=array[0..511,0..511] of real;
     picturetype=array[0..511,0..511] of byte;

var
    filename,filename2:string;
    a:real;
    p:picturetype;
    m,n:integer;
    fx,fy:smslice;
    fbr,fbi:smslice;
    d,e,f,i:fftarray;
    th,row,col,t,rho:integer;
    fsize,ifft,forward:integer;
    x,y:integer;
    sine,cosine:array[0..511] of real;

[external] procedure FFA (var y:slice;n:integer); fortran;
[external] procedure FFS (var y:slice;n:integer); fortran;
[external] procedure FFT842(var i,n:integer; var x,y:smslice);
fortran;

procedure loadimage(var image:picturetype;filename:string);

var infile:file of buffertype;
    buffer:buffertype;
    row,col:integer;
    transfer:array[0..511] of byte;
begin
   writeln('Reading ',filename);
   open(infile,filename,history:=unknown);
   reset(infile);
   for row:=0 to picsize do
     begin
```

```
        read(infile,buffer);
        unpack(buffer,image[row],0);
      end;
    close(infile);
end;



procedure saveimage(var image:picturetype;filename:string);

var p,th:integer;
    outfile:file of buffertype;
    tbuff:buffertype;
    transfer:array[0..511] of byte;

begin
  writeln('Writing ',filename);
  open(outfile,filename,history:=unknown);
  rewrite(outfile);
  for p:=0 to 511 do
    begin
      pack(image[p],0,tbuff);
      write(outfile,tbuff);
    end;
  close(outfile);
end;

procedure stretchrange(var g:fftarray;var image:picturetype);
{Stretches the dynamic range of the temporary picture}
{ store the scaled version in image }
var row,col:integer;
    max,min:real;
    b:real;
begin
  writeln('Stretching Dynamic Range ...');
  max:=g[0,0];
  min:=g[0,0];
  for row:=0 to 511 do
    for col:=0 to 511 do
      begin
        if g[row,col]>max then max:=g[row,col];
        if g[row,col]<min then min:=g[row,col];
      end;
  writeln('Range =',max-min);
  if max=min then b:=1
         else b:=255/(max-min);
  for row:=0 to 511 do
    for col:=0 to 511 do
      image[row,col]:=trunc((g[row,col]-min)*b);
end;
```

```
begin
  writeln('Load file:');
  readln(filename);
  writeln('Save file:');
  readln(filename2);
  fsize:=512;
  forward:=0;
  ifft:=1;
  loadimage(p,filename);
  writeln('1D FFT of each slice...');
  { Get 1D FFT of each slice }

  for th:=0 to 511 do
    begin
      for rho:=0 to 255 do
        begin
          fx[rho+1]:=p[rho+256,th];
          fx[rho+1+256]:=p[rho,th];
          fy[rho+1]:=0;
       fy[rho+257]:=0;
        end;
      FFT842(forward,fsize,fx,fy);

      for rho:=0 to 255 do
        begin
          d[rho +256,th]:=fx[rho+1];          { Make origin at
centre }
          d[rho     ,th]:=fx[rho+1+256];
       e[rho +256,th]:=fy[rho+1];
          e[rho     ,th]:=fy[rho+1+256];
     end;
    end;

  writeln('Zero order interpolation...');
  { Zero order interpolation }
  a:=3.1415926535/512;
  for th:=0 to 511 do
    begin
      sine[th]:=sin(th*a);
      cosine[th]:=cos(th*a);
    end;
  for y:=0 to 511 do
  for x:=0 to 511 do
    begin
      f[y,x]:=0;
      i[y,x]:=0;
    end;
  for y:=0 to 255 do
    for x:=-256 to 255 do
      begin

        rho:=trunc(sqrt(x*x+y*y));
     if x<>0 then th :=trunc((arctan(y/x)/a))
```

```
          else th:=256; {pi/2}
        if th<0   then th:=512 + th;
        if th>511 then th:=th  - 512;
        if (rho<256) and (th>0) and (th<512) then
          begin
        f[ y+256, x+256]:=d[ rho+256, th    ];
    f[-y+256,-x+256]:=d[-rho+256, th    ];
        i[ y+256, x+256]:=e[ rho+256, th    ];
    i[-y+256,-x+256]:=e[-rho+256, th    ];
          end;
     end;

writeln('2d Ifft');
 { do cols }
 writeln('Cols..');
 for n:=0 to 511 do
   begin
     for m:=0 to 511 do
       begin
         fx[m+1]:=f[m,n] * (-1)**(m+n); { shift to centre }
         fy[m+1]:=i[m,n] * (-1)**(m+n);
       end;
     FFT842(ifft,fsize,fx,fy);
     for m:=0 to 511 do
       begin
         f[m,n]:=fx[m+1];
         i[m,n]:=fy[m+1];
       end;
   end;
 { do rows}
 writeln('Rows...');
 for m:=0 to 511 do
    begin
       for n:=0 to 511 do
         begin
           fx[n+1]:=f[m,n];
             fy[n+1]:=i[m,n];
         end;
       fft842(ifft,fsize,fx,fy);
    for n:=0 to 511 do
      begin
          f[m,n]:=fx[n+1];
       i[m,n]:=fy[n+1];
      end;
    end;

 writeln('Calculating magnitude...');
 for m:=0 to 511 do
   for n:=0 to 511 do
     f[m,n]:=sqrt(f[m,n]*f[m,n]+i[m,n]*i[m,n]);
 stretchrange(f,p);
 saveimage(p,filename2);

end.
```