SPATIAL DECOMPOSITION OF THE HOUGH TRANSFORM

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

IN COMPUTER SCIENCE

UNIVERSITY OF REGINA

By

James A. Heather

Regina, Saskatchewan

February 2006

Your file Votre référence
ISBN: 978-0-494-18889-7
Our file Notre référence
ISBN: 978-0-494-18889-7

UNIVERSITY OF REGINA

FACULTY OF GRADUATE STUDIES AND RESEARCH

SUPERVISORY AND EXAMINING COMMITTEE

James A. Heather, candidate for the degree of Master of Science, has presented a thesis titled, ***Spatial Decomposition of the Hough Transform,*** in an oral examination held on December 1, 2005. The following committee members have found the thesis acceptable in form and content, and that the candidate demonstrated satisfactory knowledge of the subject material.

External Examiner:	Dr. Richard McIntosh, Department of Mathematics and Statistics
Supervisor:	Dr. Xue Dong Yang, Department of Computer Science
Committee Member:	Dr. David Gerhard, Department of Computer Science
Committee Member:	Dr. JingTao Yao, Department of Computer Science
Chair of Defense:	Dr. Gordon Huang, Faculty of Engineering

Abstract

In the field of Image Processing it is a common problem to search for the end points of lines in binary edge images generated using edge detection algorithms. Often this is done by using the Hough Transform, however in its native form the Hough Transform is only capable of finding line parameters and not the end points of those lines represented by the line parameters. In the past several enhancements to the Hough Transform have been published which deal with locating the end points associated with those parameters. This paper discusses one such enhancement which has some unique properties unlike any discussed in previous research. The enhancement involves the use of a hierarchical Hough Transform that makes use of a quad tree and an additive property such that the parameter space accumulator for a given sub-image can be stored implicitly in the tree and recovered by adding up the corresponding entries in the accumulators for each of the disjoint sub images. This paper will also discuss the trade-offs of time and space complexities as well as discuss one potential application in Augmented Reality which was the original motivation for this research. Comparisons to other hierarchical approaches to the Hough Transform will also be discussed.

Acknowledgements

I would like to acknowledge my supervisor Dr. Xue Dong Yang for his creative perspective and abstract problem solving skills that helped me see new possibilities as well as practical applications to the core idea of my thesis. His understanding and appreciation for the learning process gave him the patience to wait for me to discover things on my own, rather than merely give me step by step instructions to complete a predetermined project. His openness to new ideas and techniques gave me the freedom to think about problems from my own perspective and eventually develop the idea which became the paradigm for my thesis. He taught me how to think, rather than what to think, and that was instrumental in my education.

I would like to thank the Faculty of Graduate Studies and Research for their financial support over the past 2 years. Their support has made it easier for me to focus on my research rather than worry about how to pay my tuition fees. Their help let me spend my summers focused on solving problems related to the Hough Transform.

I would also like to thank my family for giving me a place to stay while I was getting my education. They helped encourage me to continue my education and to make the most with the skills I have. Without their support, this endeavor would have been even more difficult of an undertaking than it already was. Their guidance and support does not go unappreciated.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Overview

In the field of Image Processing the extraction of geometric features from images is a very common problem. Over the years several different approaches have been devised to extract these features. These different approaches can be characterized and classified in several different ways. Some of the techniques involve global examination of the image while others only involve local examination of each pixel in the image. Further still, some techniques involve the decomposition of the image into sub-images in an attempt to simplify the problem. Some representative examples of these different approaches will be reviewed in this thesis, particularly the issues of hierarchies and image decomposition. The discussion will be primarily limited to the extraction of lines. However, some generalizations may be made for other mathematically defined features as well.

The Hough Transform (HT) [7] is a common technique for extracting lines from images. In this thesis I will propose a variation of the standard HT called Spatial Decomposition of the Hough Transform (SDHT) [5]. The SDHT and its unique mathematical properties are the main contribution of this thesis. One of these properties, known as the additive property, will be described in greater detail in Section 3.3. This is the property that any accumulator arrays defined from a global origin which correspond

with disjoint sub-images, when their corresponding entries are added together form the accumulator array corresponding to the merged sub-image. This property has several advantages and is a significant contribution to the HT. The HT and subsequent SDHT are part of the larger problem of finding lines in digital images and therefore, some suitable background information on the subject will be given in this chapter to familiarize the reader with the topic and some similar techniques before moving on to the details of the SDHT.

1.1.1 Edge Detection

Typically the first step in the process is to perform some form of edge detection such as Canny [2] on the image, thus generating a new binary edge image that provides the necessary segmentation of the original one. Edge detection algorithms operate on the premise that each pixel in a grayscale digital image has a first derivative with regard to the change in intensity at that point. If a change occurs at a given pixel in the image, then traditionally, a black pixel (from this point on known as a feature point) is placed in the binary edge image. If a change does not occur, then a white pixel is placed there instead. In general, the gradient is compared at each pixel that gives the degree of change at each point in the image. The degree to which this derivative is considered however remains an empirical problem. The question basically amounts to how much change in the intensity should be required in order to constitute a feature point in the binary edge image. Usually a predefined threshold value T is used to classify edge points. This is to simplify the voting process which takes place in the HT. Without some form of thresholding the voting process must consider varying degrees of change not commonly dealt with in previous literature.

In an attempt to find the accurate (or optimal) location of an edge, a second derivative is often used. The edge can be found at the point where the change in the first derivative is constant, corresponding to the local maximum / minimum. This is often referred to as a zero crossing because it is the point at which the second derivative equals zero but its left and right neighbors are non-zero and have opposite signs.

It is often difficult to select the optimal threshold for mapping the grayscale image into a binary edge image of feature points. One of the primary reasons for this is due to non-uniform illumination in images. When shading cues are too subtle to pick up on, legitimate geometric features in images, primarily lines, which are the focus of this paper, are difficult to pick up on. Any attempt to fix this problem by lowering the threshold to allow for weaker lines to be detected, also tends to allow for the addition of a great deal of noise in the image. This noise hinders further steps in feature detection from finding the features of interest, rather than false positives. This problem may not seem to be significant to the reader at first glance, because of the initial desire to compromise the process by allowing it to be performed manually. This would involve letting the user run the edge detection algorithm once, and then, if the binary edge image does not properly display the features that the user deems to be of interest, run it subsequent times with varying thresholds until better results are achieved. This manual process may seem like a reasonable compromise at first, but in the case of robot vision, where the computer must take frames from video capture devices sequentially in real time and process them automatically, the compromise is of no use. Particularly in the field of Augmented Reality (AR) where time and accuracy constraints are of the utmost importance, this becomes even more of a problem. AR and its connection to image processing techniques,

particularly the method proposed in this thesis, will be discussed in greater detail in Section 6.3.

1.1.2 Problem Subdivision

In general it is common to solve complex problems by subdividing them into smaller and more manageable problems. This process reduces the complexity of the algorithms involved and increases their robustness. It would seem obvious that there could be some benefits to applying this methodology to the problem of extracting geometric features from images. There are two different ways in which this problem subdivision can be described in this problem domain. They are as follows:

1. Determine some simple (primitive) features that can be detected more easily in an image and then, based on their arrangement determine if they form a reasonable foundation for a more complicated feature.
2. Determine if a feature can be found in an image by subdividing the actual image itself such that the problem is reduced to simply checking for smaller portions of the feature in each sub-image nearby. Then it is necessary to verify if these smaller features can be legitimately combined to form a larger feature based on spatial proximity and orientation.

In order to better understand the previous two concepts consider an example related to the first concept. Suppose one wishes to find a square in an image. Finding a square in an image might be easier if one subdivides the problem into finding parallel and orthogonal lines and then using those lines to decide if a square is present in the image. Furthermore one might then use the arrangement of squares found to determine if there is

a basis for believing that a more complicated feature composed of squares is present in the image. This concept can be taken one step further by identifying the most primitive object in any image which is essentially a feature point. This general concept is the most common motivation for line finding algorithms. The assumption is that the lines have some importance given the circumstances and that they will be used for some broader and more important purpose later. The usefulness of finding lines in images will be discussed in greater detail in Section 1.4.

The second concept has been experimented with often in the past, and has been proven to enhance the process of finding geometric features in images. This concept can be used in conjunction with the first concept and therefore is a more fundamental decomposition of the problem. This thesis will discuss in detail the search for lines in images and will limit the discussion to this search. This is mainly because aside from a single point, a line is the most primitive geometric feature that can be expressed in an image. Thus, lines serve as a starting point for finding other features. The search for curves and other more complicated geometric features will be left up to the reader to research further. Some interesting techniques for finding curves in images can be found in [14] and [15].

1.1.3 Problems with the Naïve Method

There are several different methods for finding lines in images, some of which do not subdivide the problem domain at all. Therefore it is necessary to discuss these methods in greater detail highlighting their benefits and drawbacks. It is also important to consider the reason for having line finding techniques in the first place. It is important to understand why the naïve method is not sufficient for this purpose. Suppose no

consideration is given to improving the run time of this process. Then we would simply trace along the path of every possible line in the image and check for line segments that we wish to record by finding feature points within a predefined vicinity of each line. Observe the following pseudo-code for this naïve approach:

```
Begin LineSearch(LineList)
     For Each Slope 'a'
          For Each Y-Intercept 'b'
               For Each Point 'p' On The Path
                    If FirstPoint(p) Then
                         p0 = p
                    Else If LastPoint(p) Then
                         p1 = p
                         LineList.AddLine(p0, p1)
                    End If
               End For
          End For
     End For
End LineSearch
```

Figure 1.1 Pseudo-code for the naïve approach

It can be seen that the run time characterization of such an algorithm would be $O(n^3)$ and prohibitive for all but the smallest of images. There is another problem with this approach. If a pixel in the image does not represent a feature point, then it will still be processed again for any other lines passing through it, even though it has already been established as being empty space in the image. Also it is not uncommon in line searching algorithms to perform some additional line verification processing. This is to consider the surrounding area of the line and decide if the line is in fact a line, rather than just an arbitrary collection of feature points in the image. Basically for each segment of feature points along the path, it is necessary to traverse all of the orthogonal points within a specified interval in order to perform line verification. If this is done, then the

neighboring lines will also be visited simultaneously. This means that every line will be visited several times depending on the width of the point interval chosen. In the next section, a more efficient approach to this problem known as the HT will be discussed. This is the approach that will serve as the basis for the concept discussed in this paper.

1.2 The Hough Transform

The HT is a robust method for finding lines in images that was discovered by Paul Hough. In 1952 Donald A. Glazer invented the bubble chamber for which he later was awarded the 1960 Nobel Prize in Physics. A bubble chamber is a cylinder filled with super heated transparent liquid which is surrounded by a constant magnetic field. Charged particles in the chamber nucleate bubbles of vaporized liquid and thus leave a helical trail behind them. A camera placed at the top of the chamber records the scene. Although Hough's 1962 patent [7] was originally filed for the purpose of detecting the paths of particles in the above mentioned bubble chamber images, it was later found to be useful for solving many other problems involving the search for lines in images as well.

The main premise for the HT is as follows. For each line L, there exists a unique line L', which is perpendicular to L and passes through the origin. L' has a unique distance ρ and angle θ from the horizontal axis of the image. This angle and distance define a point in parameter space sometimes known as Hough space. A point in image space has an infinite number of lines that could pass through it, each with a unique ρ and θ. This set of lines corresponds to a sinusoidal function in parameter space. Two points on a line in image space correspond to two sinusoids which cross at a point in parameter space. That point in parameter space corresponds to that line in image space, and all sinusoids corresponding to points on that line will pass through that point.

The computer is incapable of checking an infinite set of lines because the algorithm would never end. Also, even if this were possible it would not be desirable because the image is digital, and therefore has a limited resolution not capable of representing differences in the orientation of lines beyond a certain degree of accuracy. This topic of line parameter accuracy will be discussed in Section 4.2. The solution to implementation is to quantize the parameter space by using a 2D array of counters where the array coordinates represent the parameters of the line. This is commonly known as an accumulator array. In this thesis, an element in this array will be referred to as a houghel, which is short for "Hough Element." The HT method for finding lines in images generally consists of the following three stages:

1. Perform accumulation on the accumulator array using the binary edge image.
2. Find peak values in the accumulator array.
3. Verify that the peaks found correspond to legitimate lines rather than noise.

The 3rd step mentioned is not entirely necessary depending on the circumstances, however false detections of lines are not uncommon when using the HT. In Hough's original patent the line equation used to parameterize the lines was Equation 1.1. The unbounded nature of this equation combined with the not only possible but high likelihood of vertical lines in images made this scheme for parameterization rather flawed. In 1972 however Duda and Hart published [3] which described Equation 1.2 as being an unbounded parameterization for solving the problem. They described this as being the General Hough Transform (GHT) and this is the HT used most commonly today. The 1st step mentioned is the most important step because it reduces the problem to

finding peak values in an array. The pseudo-code for the HT accumulation process is as follows:

```
Begin HoughTransform(Accumulator)
     For Every Feature Point (x, y)
          For Every Gradient Angle θ
               ρ = y*sin(θ) + x*cos(θ)
               Quantize(ρ, θ)
               ++Accumulator[ρ, θ]
          End For
     End For
End HoughTransform
```

Figure 1.2 Pseudo-code for the Hough Transform

The two different formulas for the parameterization of a line are as follows:

$$y = a * x + b \qquad (1.1)$$

$$\rho = x * \cos(\theta) + y * \sin(\theta) \qquad (1.2)$$

The HT has several benefits over the naïve approach to finding lines in images. The list of benefits is as follows:

1. The HT has a general run time characterization of $O(n^2)$.
2. The HT is not sensitive to gaps in line segments as a result of fuzzy images.
3. The HT does not revisit feature points.
4. The HT is capable of culling non-feature points from the problem space entirely.

Unfortunately, the second advantage in the list above is closely related to the first disadvantage in the list below. The list of limitations to the HT are as follows:

1. The HT does not consider the connectivity of feature points when voting in the parameter space and therefore does not record the end points of lines.
2. The parameter space often has false peaks as a result of noise in images combined with other lines that intersect the line in question.

1.3 Non-Hierarchical Methods for Finding Lines

Not all methods for finding lines in images are hierarchical methods. Many methods do not even make use of the HT. One such method discussed in [18] uses a polygonal approach where the outline of geometric features is obtained and afterwards a best-fit algorithm is applied to see if a line passes adequately through the convex hull formed by the feature points. This method is reported to be fast and efficient but is for the purposes of finding the exact end points of the lines in the image with precision. It should not be used in situations where only general information about line location is required.

Another method for accomplishing this which will be discussed in further detail in Section 5.3 is [20]. This method employs the HT using a single accumulator array on a global scale but does so by recording the min and max range of the feature points that voted on each parameter pair. Thus, the following structure for an accumulator element also known as a houghel is used instead of the traditional element.

```
typedef struct Houghel
{
     unsigned short total_votes;
     unsigned short min_range;
     unsigned short max_range;
} *LPHoughel;
```

Figure 1.3 C/C++ Houghel definition

Using this new structure at each point in the accumulator array allows for the ability to record the range of the line. Due to the problem explained earlier, in some houghels it is necessary to record the min and max range of the x component, while in others the min and max range of the y component, because there may be some vertical or

horizontal lines. The value of the angle of the line becomes necessary to distinguish which is being recorded when each vote is cast. This is not a significant problem however and leads to a rather efficient solution. After the accumulator array has been voted on the interval between the minimum and maximum values in the range is checked.

1.4 Motivation of the Current Research

Finding lines in images has several purposes for many different applications. They can often be used to find objects on the ground in satellite images. In the case of [16] road segments were extracted from images for the purposes of updating Geographic Information Systems (GIS) devices. In [11] the extraction of lines aids in the extraction of culturally important architecture which can then be stored in the computer as a fully 3D textured model. This concept in particular is similar to one that is of importance in the research in this article because it involves the extraction of 2D lines which are then mapped into 3D space and used to model the structure found in the image. In [22] palm lines are extracted in order to perform the recognition of palm prints. Similarly, in [24] the HT is used to extract thumb prints and even perform iris recognition. In [23] the lines are extracted from form documents, often with complicated backgrounds that make their extraction more difficult.

It would seem readily apparent that the ability to find lines in images is advantageous to solving many detailed problems. The HT as described in Section 1.2 is a sophisticated global technique for finding the parameters of lines in images but lacks the ability to find the end points of those lines. In previous literature hierarchical techniques have been used to subdivide a digital image for the purposes of devising the locality of the end points found using the HT. Much of the research devoted to this concept however

was done in the 1980's when memory was in short order and therefore suggesting a technique that would require a larger amount of it was unthinkable. Due to this limitation every technique always began with the use of a smaller accumulator derived from a local coordinate system at the top left hand corner of each sub-image. Almost all other variations after that point were limited to the type of decomposition and the form of the hierarchy. The techniques required numerous passes to the HT system and suffered from the problem of having to always map line parameters from local coordinate systems back to the global coordinate system in order to match line segments up with each other, if so required. Therefore the SDHT was devised as a new technique for modern computers which is capable of solving these problems with the trade off of higher memory requirements. It is in the opinion of the author that this particular research was not possible during the time in which this kind of research was originally being conducted. Therefore, it is worth a second look, now that significant changes have taken place in the computer industry.

To summarize, the main concept which will be discussed in the remaining chapters is the decomposition of binary edge images into disjoint sub-images each with their own parameter space defined from the global origin of the original image. The use of a quad-tree in the discussed implementation allows for the implicit storage of quantized accumulator arrays representing parameter spaces at each level of decomposition. The use of a global origin for all parameter spaces reduces the problem of grouping co-linear line segments to a function of proximity in parameter space.

The main contributions of this thesis are listed here:

1. A simple and mathematically elegant solution is devised for the spatial decomposition of the image which is easier to implement than data structures such as pyramids devised in previous research papers.
2. The property that each level of a quad-tree implicitly stores the parameter spaces at all previous levels of the tree.
3. The property that grouping co-linear line segments found in the image may be simplified to the process of grouping all line segments found within a predetermined proximity of each other in parameter space.
4. The fact that the modified HT algorithm unlike its predecessors is a one pass algorithm, only needing to perform the accumulation process once.

1.5 Thesis Outline

In the first chapter we have examined the general problem of finding lines in images as well as several different approaches and why the HT is advantageous to solving the problem. The motivation for the SDHT technique has been addressed and so Chapter 2 will go into greater details with various adaptations to the HT and in particular put focus on the pros and the cons of pyramid hierarchies used to solve the problem as they are the most relevant to this topic of research. In Chapter 3 the properties of the SDHT and their advantages as well as disadvantages will be discussed. Implementation details as well as resulting images will be included as well. In Chapter 4 the issues of spatial resolution for both the global image as well as the sub-images will be covered. In Chapter 5 the parameterization of the accumulator arrays and the problems entailed there will be discussed in detail, particularly with new problems raised with the SDHT.

Chapter 5 will also discuss the issue of how to choose the range constraints of the continuous parameters which can be found in the image. In Chapter 6 a sample problem in Augmented Reality (AR) will be discussed as well as how the SDHT applies directly to it. Chapter 7 will conclude with a summary of the discoveries made as well as some suggestions for future research in the area.

Chapter 2

Traditional Spatial Decomposition Techniques

2.1 Introduction

In the previous chapter we discussed the general problem of finding lines in images and more particularly the benefits of the HT. Although we discussed the possible usefulness of using spatial decomposition as a way of breaking this general problem into a series of sub problems we did not give any specific examples of how this was done in the past. Past research attempts have commonly experimented with a paradigm known as the pyramid method. This method, while different from the SDHT, has several similarities, and therefore needs to be examined in terms of its advantages and disadvantages in order to better understand the general problems associated with the SDHT.

Quad-trees and pyramid structures are used for many purposes in both the field of image processing as well as in the field of computer graphics. Naturally it would make sense to experiment with these data structures to see what possible benefits they could have in this specific problem domain. Pyramid structures, particularly the one used in [6, 17, 19], represent overlapping sub-images. Thus, they are capable of offering refinement in a manner that the SDHT is not. The SDHT however as we shall see in the next chapter is capable of storing the necessary information for refinement implicitly. This is an

interesting trade-off between these two ideas that is certainly worth looking at. Another trade-off has to do with the actual accumulation process associated with each structure.

With a pyramid structure there must be a separate accumulation process for each sub-image represented in the pyramid. Since many of the sub-images overlap this means that several feature points are accumulated multiple times by the algorithm. In a typical quad-tree this is not the case because each sub-image is disjoint from all other sub-images. Therefore each feature point need only be accumulated once to achieve the desired result. Pyramid methods are also somewhat more complicated to program than quad-trees and in computer science in general simple and elegant solutions are usually preferable over intricate and complex ones. In Sections 2.2 and 2.3 the structure of the pyramid and the structure of the quad-tree are reviewed in detail.

2.2 The Structure of the Pyramid

In [6, 17, 19], the pyramid structure is actually defined as being a hierarchy in which each level of the hierarchy consists of a number of 4x4 sub-images of pixels, who's 2x2 central sub-images of pixels composes the next higher level in the structure (Figure 2.1). The neighboring sub-images are also considered when grouping lines at higher levels. Each level of the pyramid represents the entire image broken down into a different number of sub-images. Given P + 1 levels, each level L, contains $2^{P-L} \times 2^{P-L}$ sub-images. The lowest level of the pyramid, level 0, represents the finest subdivision. The highest level of the pyramid denoted by P, represents the entire image without any subdivision. At each level of the pyramid the number of sub-images decreases by a factor of 4. So if there are N sub-images at level L then level L – 1 must have N / 4 sub-images. The HT is called once for every sub-image and then these lines

found are matched up with lines found in larger overlapping sub-images at the next level of the pyramid. Any of the same lines found are propagated to a higher level in the hierarchy. The process is then repeated until the top of the pyramid is reached. Naturally, as the level in the pyramid increases the HT accumulator arrays must increase in size as well in order to deal with increased spatial dimensions of the images. Due to this structure it is impossible to connect lines in different sub-images without performing necessary calculations to map them to a common global coordinate system. This is because every accumulator array is defined from a different local origin than that of its neighbors. Thus, although smaller accumulators can be used for smaller sub-images, eventually the entire image must be processed anyway and some time must still be lost when performing the verification necessary to determine connectivity and co-linearity between line segments in neighboring sub-images.

One of the benefits to this structure is that sometimes, when dealing with smaller images, it becomes increasingly easy for slight variations and noise in two short co-linear line segments, part of a larger line segment, to become decreasingly co-linear due to their partitioning. When this occurs they may still be grouped together within reason using the quad-tree family of implementations, however it becomes increasingly difficult and usually similarity thresholds with regard to their co-linearity must be relaxed to allow for this. With the pyramid methods however there is the use of overlapping sub-images and therefore, information is gained more than once for any given set of feature points lying in these regions. This is because the problem of having a precise and steadfast partition has been resolved by having a softer boundary that reflects the spatial connectivity needed to overcome this problem. The reused feature pixels help to alleviate this issue

while introducing the issue of increased run time as mentioned in the previous section. This reuse of feature points should not be confused with a change in pixel resolution at higher levels of the pyramid. Each sub-image still contains the same pixel resolution. Only the dimensions of the sub-images change at higher levels. The reason for the trapezoid shape of Figure 2.1 is because lines found in the central sub-images, (shown in black), are the only ones considered in the larger sub-image directly above at the next highest level. The neighboring sub-images are merely used to gather further evidence for line segments already found in the central sub-images, if they contain co-linear line segments. This is how the reuse of feature points is achieved because the neighboring sub-image at level $L + 1$ will also acquire evidence from some of these neighboring sub-images at level L. Level $L + 1$ does not represent level L at a slightly smaller pixel resolution.

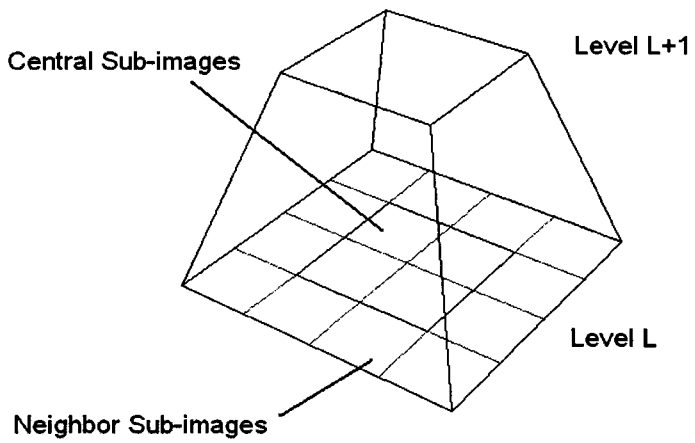


Figure 2.1 The Pyramid Structure

2.3 The Structure of the Quad-Tree

The quad-tree structure in this problem domain is defined as being a hierarchy in which an image can be represented as a series of disjoint sub-images. Each level of the quad-tree represents four sub-images in the global image and each of those sub-images may be decomposed even further into four more sub-images recursively until the sizes of the sub-images have reached a desired minimum (Figure 2.2). This representation allows for the problems discussed above but also allows for some interesting properties depending on the implementation. Due to the fact that each sub-image (assuming the tree has sufficient levels) is relatively small, smaller accumulators may be used as is the case with the pyramid structures. Also similar to the pyramid structures is the fact that each accumulator is derived from a local origin at the top left corner of the sub-image to which it is associated with. Therefore the problem of determining co-linearity is still present. However, due to the disjoint representation provided by the quad-tree, information found at higher levels can be found implicitly assuming that a slightly different technique is used. The SDHT, which is a member of the quad-tree family of implementations, was described in Section 2.1 as having this particular property and as being able to acquire information at higher levels for the purposes of refinement in line segment detection. Unlike other quad-tree methods whose main advantage is the use of smaller accumulator arrays for the purposes of saving on accumulation time, the SDHT sets out to abandon this advantage in the hopes of opening up a whole new set of properties which will allow for a greater understanding of the line segments detected in the image depending on the implementation. The SDHT defines all of its accumulators from the global origin in an attempt to simplify the grouping process required to map short line segments to longer

line segments. The quad-tree structure when used in this way supports efficient computation but with an increased cost in memory. Due to increases in memory storage by modern computers, as is discussed in Chapter 5, the ability to find line segments accurately within a reasonable time frame is preserved. Another slightly different approach to using a quad-tree in conjunction with the HT for the purposes of decomposition can be found in [12]. Now that we are familiar with the differences in the two most common approaches used to structure this problem with, we will make use of the next chapter to discuss the finer details of the SDHT and its properties.

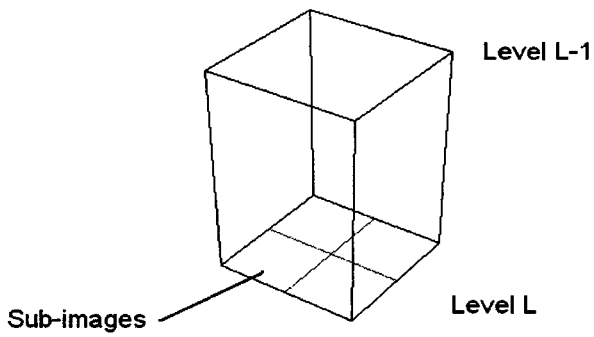


Figure 2.2 The Quad-Tree Structure

Chapter 3

Properties and Techniques

3.1 Understanding the Problem

In the previous chapter several different adaptations to the HT were discussed. Some of these adaptations were hierarchical and involved the decomposition of the image space. The focus of this paper as discussed in the first chapter is to describe a new method for decomposing image space in an attempt to localize image features such as lines, and reduce the complexity of the problem. We would like to find the end points of line segments in binary edge images. However, many techniques do not allow for this. In papers [9, 10] it is possible to take a parameter space and map it perfectly back to the original image. This however, still does not provide information about the line segment end points. Previous methods discussed took advantage of the smaller resolution of sub-images by using smaller accumulators, and thus, improving the time needed to perform accumulation. Several problems were introduced by this technique that will be addressed by the SDHT. The additive property of the SDHT serves as the foundation for the technique, making it possible to overcome some of the drawbacks to previous hierarchical methods and will be discussed in detail in this chapter.

The pyramid methods for spatial decomposition regardless of their implementation always suffer from the same problem of having to re-accumulate overlapping sub-regions. Even if this accumulation is fairly limited it still exists as a

drawback to the method and requires a second look. Suppose we wish to have information about the decomposition of the image space at different levels of resolution. Given this scenario we would have to re-accumulate several times at different levels in a similar way to the pyramid methods. This is obviously undesirable and we wish to find a way to accumulate at all levels simultaneously. If only one accumulation is performed however, then only one global accumulator is acquired and local information about the locations of the image features is lost during the voting process.

3.2 Quad-Tree Implementation

The problem discussed above arises from the lack of localized information about the sample points being voted on in parameter space. This information which is known at the time of the voting process is lost thereafter. The key to the method proposed here is to preserve this knowledge during the voting process. This causes only a slight increase in complexity and memory space but allows for a more detailed line segment extraction process later on. The vote is recursively propagated in a quad-tree whose definition is such that the root node represents the full image and each of the four sub-trees represents a quadrant of the parent node for each node in the quad-tree.

The quad-tree, used to represent the image, recursively segments it into four sub-images, each of which has its own corresponding parameter space as defined from the global origin rather than a local one. A vote for a feature point is only made to a leaf node whose parameter space corresponds to the spatial location of the image block which contains the feature point in question. Each leaf node represents the parameter space manifold of a disjoint subset of feature points within the global set of image points *I*. Thus, the intersection of these parameter space subsets is the empty set. In addition to the

parameter space represented, the total number of feature points located within each of these disjoint subsets is also recorded in each leaf node. This value however, as it is propagated down the quad-tree is added to the total of each branch node on the way down. This leads to the property that the total feature count for each quad-tree is equal to the sum of the feature counts of all four sub-trees. This allows for the pruning of sub-trees falling below a specified threshold and thus representing image regions largely unpopulated by feature points.

3.3 The Additive Property

In performing the localized voting process one of the main concerns is the availability of global peak information potentially lost as a consequence of localized information. The local quad-tree has an additive property however which allows for the acquisition of the original global parameter space manifold. This property allows for the arbitrary acquisition of the parameter space manifold for any disjoint quadrant subset of feature points in the global image I_{NxN}. Thus, it is possible for adjustable refinement of the spatial resolution of the line segment end points. The additive property asserts that the quantized parameter space of each quad-tree is equal to the summation of that of its sub-tree parameter spaces. Let Q represent an arbitrary non-leaf node of a quad-tree in the data structure whose implicit parameter space manifold A, corresponds to a disjoint quadrant of M^2 points J_{MxM} such that $J_{M\times M} \subseteq I_{N\times N}$. Let Q_{nw}, Q_{ne}, Q_{sw} and Q_{se} represent the northwest, northeast, southwest and southeast sub-trees of Q respectively. Where F represents the total number of feature points found within J_{MxM}, let F_{nw}, F_{ne}, F_{sw} and F_{se} represent the number of feature points found within the four disjoint quadrants of J_{MxM}. As well, let A_{nw}, A_{ne}, A_{sw} and A_{se} represent their respective parameter space manifolds.

Then Equations 3.1, 3.2 and 3.3 are true by definition. Equation 3.4 subsequently holds for any quad-tree Q and the higher resolution information can be obtained through addition. Let the $\oplus$ operator denote the addition of every corresponding element in two parameter space manifolds. For the results of Equations 3.2, 3.3 and 3.4, see Figure 3.1, which represents the subdivided parameter space of the edge image found in Figure 3.5b.

$$\forall Q \mid Q_{nw}, Q_{ne}, Q_{sw}, Q_{se} \subseteq Q \Rightarrow F_{nw} + F_{ne} + F_{sw} + F_{se} = F \quad (3.1)$$

$$\forall Q \mid Q_{nw}, Q_{ne}, Q_{sw}, Q_{se} \subseteq Q \Rightarrow Q_{nw} \cap Q_{ne} \cap Q_{sw} \cap Q_{se} \equiv \varnothing \quad (3.2)$$

$$\forall Q \mid Q_{nw}, Q_{ne}, Q_{sw}, Q_{se} \subseteq Q \Rightarrow Q_{nw} \cup Q_{ne} \cup Q_{sw} \cup Q_{se} \equiv Q \quad (3.4)$$

$$\forall Q \mid Q_{nw}, Q_{ne}, Q_{sw}, Q_{se} \subseteq Q \Rightarrow A_{nw} \oplus A_{ne} \oplus A_{sw} \oplus A_{se} \equiv A \quad (3.5)$$

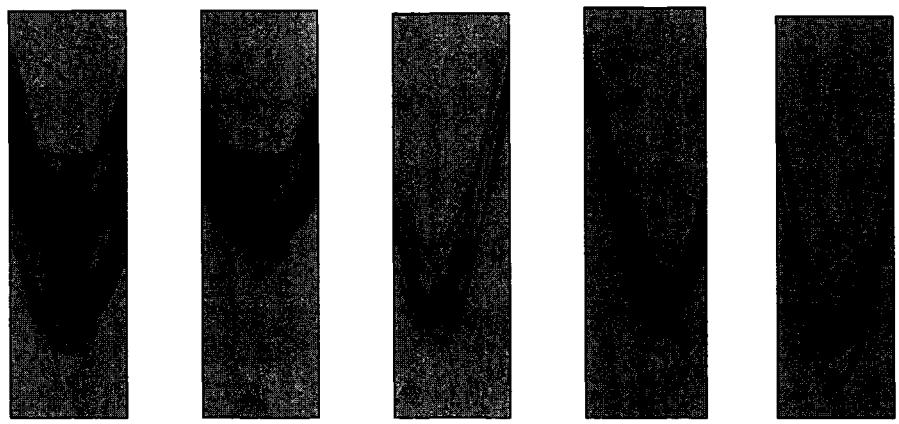


Figure 3.1 Accumulators (a) Full Image; (b) Top Left Sub-Image; (c) Top Right Sub-Image; (d) Bottom Left Sub-Image; (e) Bottom Right Sub-Image

In order to create a quad-tree structure to contain the localized parameter space manifolds needed for spatial information, there is a tradeoff which needs to be discussed with regard to the degree of memory required to create the tree, as well as to acquire adequate resolution for the line segments of the image in question. The accumulator arrays representing the parameter space manifolds must be composed of two byte integers

in order to sufficiently hold votes large enough to cover most image sizes. There are two ways to implement the tree structure depending on the trade-offs deemed acceptable by the user. One method is to allow only the leaf nodes of the quad-tree to contain accumulator arrays. This means that the accumulator arrays at any other sub-tree in the structure are purely implicit and must be derived using the additive property. This is the implementation discussed in this paper and of greater efficiency in cases where the same parameter space data will not be used repeatedly or propagated back up the hierarchy in an attempt to generate groupings at higher levels as in [17]. Although, if implicit storage is used, efficient hierarchical line grouping can still be achieved by only applying the additive property to accumulator cells which are, or are near to peak values within the parameter space at the local level. The results shown in this paper did not demonstrate line groupings but this certainly can be achieved. There are two formulas for representing this storage method. The first is in terms of block sizes and the second is in terms of tree depth. Let each sub-image J_{MxM} of the image I_{NxN} have a block size of M^2. Let d denote the maximum depth of the quad-tree from the root, such that the depth at the root is zero. Let $Total_{imp}$ denote the total number of accumulator arrays needed to represent each leaf node of the quad-tree, thus, implicitly storing accumulator arrays at arbitrary quad-tree depths. Let $Total_{exp}$ denote the total number of accumulator arrays needed to represent every node of the quad-tree at all depths, thus, storing the arrays explicitly. Equation 3.5 holds in the former case and Equation 3.6 holds in the latter.

$$\left(\frac{N}{M}\right)^2 = 4^d = Total_{imp} \qquad (3.5)$$

$$\sum_{\beta=0}^{d} 4^{\beta} = Total_{exp} \qquad (3.6)$$

It can be noted that the above additive property is held for arbitrarily shaped disjoint image subdivisions.

One of the benefits of the decomposition of the HT is that it does not introduce a significant penalty in the overall run time characterization of the algorithm due to the fact that a quad-tree is being used for the voting process. This means that if the total number of feature points is u, the total number of quantized units of θ is v, and the total number of sub-images is w, then the run time characterization of the voting process would be $O(\log_4 w^{uv}) = O(uv)$. The pseudo-code can be seen in Figure 3.2.

The implementation described has one slight limitation with the run time of the voting process that is caused by the propagation of votes down the quad-tree with each new feature point being processed. This can be improved with a slightly modified accumulation algorithm. The new algorithm involves the complete traversal of the quad-tree to arrive at each leaf node. At this point, accumulation on all of the feature points in the corresponding sub-image $J_{M\times M}$ is performed. With this slight modification to the algorithm the total features denoted by F are recorded at the leaf node, and then returned to the parent tree so that all levels of the tree retain information about the feature count. The pseudo-code can be seen in Figure 3.3.

```
Begin GetLocalAccumulator(x, y, Q)
     ++F
     If IsLeafNode(Q) then
          Return A
     Else
          Return GetLocalAccumulator(x, y, Q_child)
     End If
End GetLocalAccumulator

Begin Accumulate(I_NxN, Q)
     For each feature point (x, y) in I_NxN do
          A = Q.GetLocalAccumulator(x, y)
          For (θ = θ_min; θ <= θ_max; θ += dθ)
               ρ = x*cos(θ) + y*sin(θ)
               Quantize(ρ, θ)
               A.Vote(ρ, θ)
          End For
     End For
End Accumulate
```

Figure 3.2 Top down SDHT accumulation pseudo-code

```
Begin Accumulate(J_MxM, Q)
     If IsLeafNode(Q) then
          For each feature point (x, y) in J_MxM do
               ++F
               For (θ = θ_min; θ <= θ_max; θ += dθ)
                    ρ = x*cos(θ) + y*sin(θ)
                    Quantize(ρ, θ)
                    A.Vote(ρ, θ)
               End For
          End For
     Else
          F_nw = Accumulate(J_MxM, Q_nw)
          F_ne = Accumulate(J_MxM, Q_ne)
          F_sw = Accumulate(J_MxM, Q_sw)
          F_se = Accumulate(J_MxM, Q_se)
          F = F_nw + F_ne + F_sw + F_se
     End If
     Return F
End Accumulate
```

Figure 3.3 Bottom up SDHT accumulation pseudo-code

The actual quad-tree used in these algorithms in the C++ programming language is shown below and a more detailed examination is given for each data member of the tree. This structure is designed to allow for a perfect sub-division of the image into sub-images of equal resolution. The spatial block boundaries are stored in the structure because they take very little memory and allow for some faster processing when selecting feature points within the image sub-region.

```
typedef struct Tree
{
     ParameterSpace parameter_space;
     Block          spatial_block;
     PeakList       peak_list;
     int            feature_count;
     Tree*          trees[4];
} *LPTree;
```

Figure 3.4 C/C++ quad-tree definition for the SDHT

The above structure is merely a suggestion for the general case of recursive image subdivision. It does not necessarily provide the optimal solution for all problems that can be encountered by the SDHT in image processing. This will become more apparent in Chapter 6 when applications in augmented reality are discussed in full. Another issue related to parameter spaces is their potential storage on the hard drive. Suppose one wishes to perform the SDHT but not immediately process the accumulator arrays generated. The accumulators, each corresponding to a different sub-image can be stored on the hard drive but at a somewhat high cost, due to their size. The SDHT contains an accumulator for each sub-region that is the same size used for the global image. It can be noted that there are several lines which could never possibly intersect a given sub-image, which are represented in the quantization of the parameter space. This is a significant

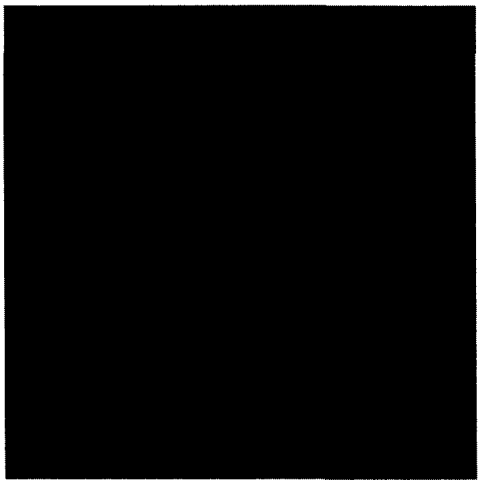
problem with the SDHT that will be discussed in greater detail in the next chapter. In terms of accumulator storage, this is not a significant problem, because empty accumulator entries, houghels, are represented with zeros and therefore represent redundancy that is easily dealt with by run length encoding and various other compression techniques.

In addition to this form of compression one might see the potential for another type of compression which is the implicit storage of parameter spaces. Since we have already shown by the additive property that parameter spaces at higher levels in the quad-tree are stored implicitly, it stands to reason that if the quad-trees at lower levels in the quad-tree are saved to the hard drive, then these higher level quad-trees are stored there implicitly as well. One may see another potential form of compression though with the additive property and this is with the storage of accumulator arrays related to overlapping sub-images. This would seem like a possibility at first but unfortunately, this is not the case. If two overlapping sub-images are accumulated separately then the sub-image that is created by their overlap and the two sub-images that are created by their subtraction from each other in set theory cannot be recovered. This is due to the uncertainty involved with determining how many votes in the overlapped sub-image accumulator can be attributed to each of the other sub-images. So this property is not something that can be achieved with the SDHT. Therefore, Equations 3.2, 3.3, and 3.4 are not just mere suggestions for the additive property, but are steadfast rules, never to be broken.

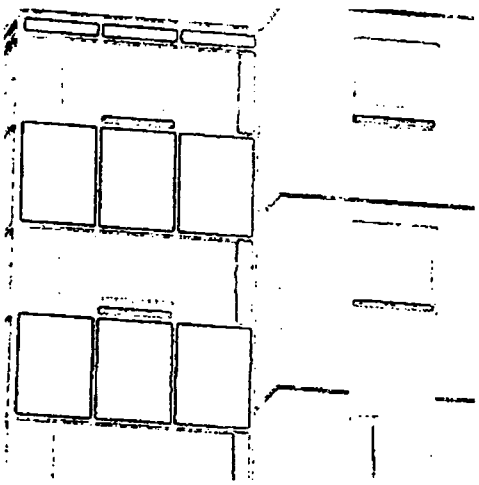
Figures 3.5, and 3.6 are digital images where $N = 512$ and $M = 32$. Therefore, by Equation 3.5, $d = 4$. As can be seen, not all lines were perfectly detected due to thresholding issues, which will be discussed in Chapter 5. The lines in Figures 3.5c, and

3.6c were acquired separately and not grouped together in any way. There is an important issue involved with this however because there are different approaches that could possibly be used when grouping line segments. Any line of feature points passing through several sub-images, when subdivided into smaller line segments tends to become fragmented and discontinuous. This relates to the issue of edge detection, as was discussed in the introduction of this paper. Figure 3.6a was provided courtesy of [4].

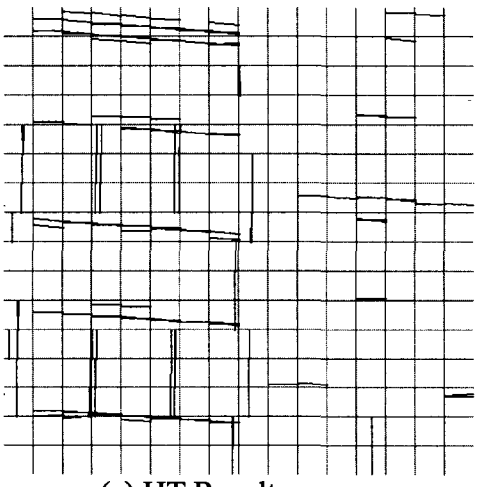
The lines, due to the noise and imperfection associated with edge images, tend to be found at slightly different orientations. In past image space subdivision methods, such as the pyramid implementations, there was a problem with grouping line segments into larger lines because each parameter space associated with a sub-image was defined from a local coordinate system and therefore required some transformations in order to get the lines in the same coordinate system again to group them. With the SDHT however, this is not the case because each accumulator array is defined from the global coordinate system so line segments can be matched up quite easily by comparison.



(a) Original Image



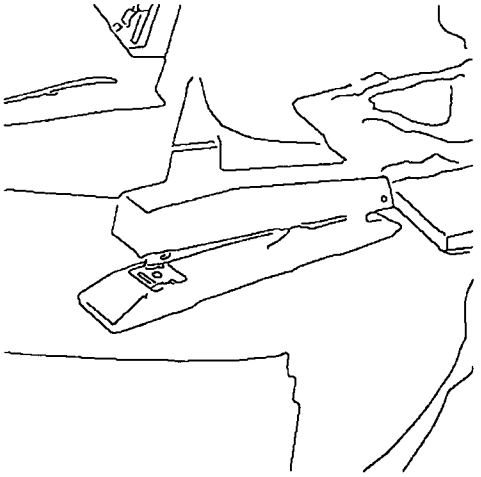
(b) Edge Detected Image



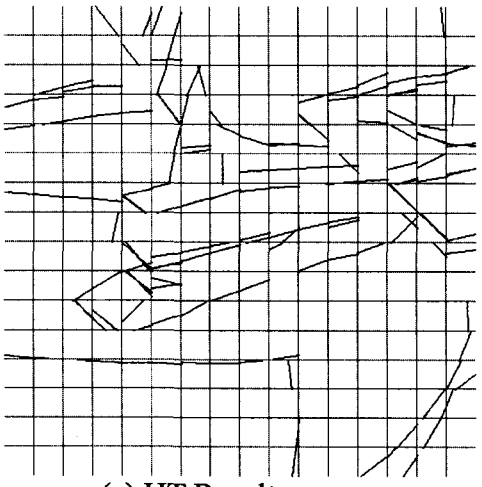
(c) HT Results

Figure 3.5 The building example.

(a) Original Image



(b) Edge Detected Image



(c) HT Results

Figure 3.6 The stapler example.

From a theoretical standpoint, by the additive property if a line is found at a given point in the parameter space of a sub-image, then that same line should be found at the same location in the parent sub-image. Due to the quantized nature of both the digital edge image and the parameter space accumulator array, this property rarely holds in a real world implementation and the peak is usually found somewhere nearby instead. Due to this nature it becomes necessary to refine the process for grouping line segments. So suppose we have two lines, denoted by *L0* and *L1*, represented by peak values, each found in different accumulator arrays from each other, which were derived from disjoint sub-images. Assume that both *L0* and *L1* each have their own end points *p0* and *p1*.

Now with this in mind, to refine the process we simply perform the following two steps to decide if *L0* and *L1* should be grouped together into a longer line segment in image space:

1. Verify that *L0* and *L1* can be found within a given minimum proximity from each other in parameter space
2. Verify that the end points of *L0* and *L1* are within a given minimum proximity from each other in image space.

If both of these steps were accomplished then the simple merging of the end points *p0* and *p1* that were in close proximity to each other, as verified in the second step, is all that is required to combine the two line segments into a larger line segment. The problem with this method, however, is that it does not necessarily reflect the line segment of best fit to the feature points that it was derived from originally. In order to accomplish this, we need to remember that for each line segment that composes the larger line

segment there is a disjoint sub-image associated with it. Observe the added information which can be derived from Equation 3.7 given that μ is the total number of line segments contributing to the larger line segment and J_i is a given sub-image with a corresponding accumulator array A_i, where a_i is a given peak entry (i.e. houghel) located in A_i.

$$\bigcup_{i=1}^{\mu} J_i = J' \Rightarrow \sum_{i=1}^{\mu} a_i = a' \mid a_i \in A_i \wedge a' \in A' \qquad (3.7)$$

Given that this is the case, we do not need to re-accumulate at all. We only need to add together the corresponding accumulator arrays involved and search for the peaks again. The accumulators are rather large so we need only add the houghels together that are in close proximity to both *L0* and *L1*. By doing this, in virtually any situation we may simply find the maximum peak in the region which is generally quite efficient in terms of run time. Most likely though, this added accuracy is not particularly needed and therefore not particularly worth the added processing required to achieve it. This can be seen by observing the results in Figure 4a. In addition, there is the possible problem that if there are two lines which both occupy the same given set of disjoint sub-images, (most likely parallel lines), and one is slightly longer than the other, then the shorter line will never be found by adding together the corresponding accumulator arrays and finding the maximum peak value because the value for the longer line will always be returned. Even if only the houghels in close proximity are added together there is still the chance that the lines are close parallel lines, (which is quite common in many man made structures), and the approach will still fail.

Chapter 4

Spatial Resolution Issues

4.1 Relationships between N and M

When considering the effectiveness of any extension to the HT such as the SDHT it is necessary to consider the spatial resolution of the image being processed. The larger the size of an image N is, the more feature points there are likely to be and the longer the processing will take. Another consideration is M, the size of the parameter space accumulator array which must also increase in size to accommodate a larger image. The full continuous range of the gradient angle must remain the same but the step size must decrease in order to maintain a reasonable degree of accuracy. The retina size must increase as well and so the continuous range of the gradient length increases, yet the step size remains the same. These observations are of great importance because unlike all other methods using the HT, the SDHT uses an accumulator large enough to sustain accuracy for the whole image with each sub-image.

The traversal of such large accumulators can cause problems with the run time of the algorithm. This is one of the reasons why modern researchers often times hesitate to use any form of the HT at all when searching for lines in images. Generally, the HT is reserved for smaller images and is not tested on larger ones. Another important issue regarding the spatial resolution of the image is the issue of image subdivision. Suppose a constant value of M is chosen for the block size that will be used with the image. If N is

the size of the image, then the larger the value of N, the more subdivisions it will take to achieve the desired sub-image size of M. Another important point to note is that as N increases, d, (the depth of the tree structure), increases at a fairly moderate rate. However, just one increase in d results in a multiplication of the total number of sub-images by four. Therefore, it is possible to actually run out of memory even with the most modern of computers because each sub-image requires such a large accumulator array. This may seem like a significant problem but in reality it does not rule out the possibility of adding up sub-regions of the accumulators in order to refine line parameters. The reason for this is that each accumulator can be created, searched, and then dispensed of individually, without having to have them all exist in memory at the same time. If a patch of houghels in the accumulator needs to be saved then this is still not a significant problem because most sub-images will not have more than one or two lines anyway, at the lowest level of the quad-tree, and therefore, there is very little storage space required even in rather large images. Typically, the only time several lines can be found in the same sub-image at the lowest level of the quad-tree is when several lines intersect with each other to form the corner of some man made structure displayed in the digital image. In cases like this if the sub-images are sufficiently small, such as a block size of $M = 8$, then not enough information is lost to constitute a problem in anything but the most stringent of applications.

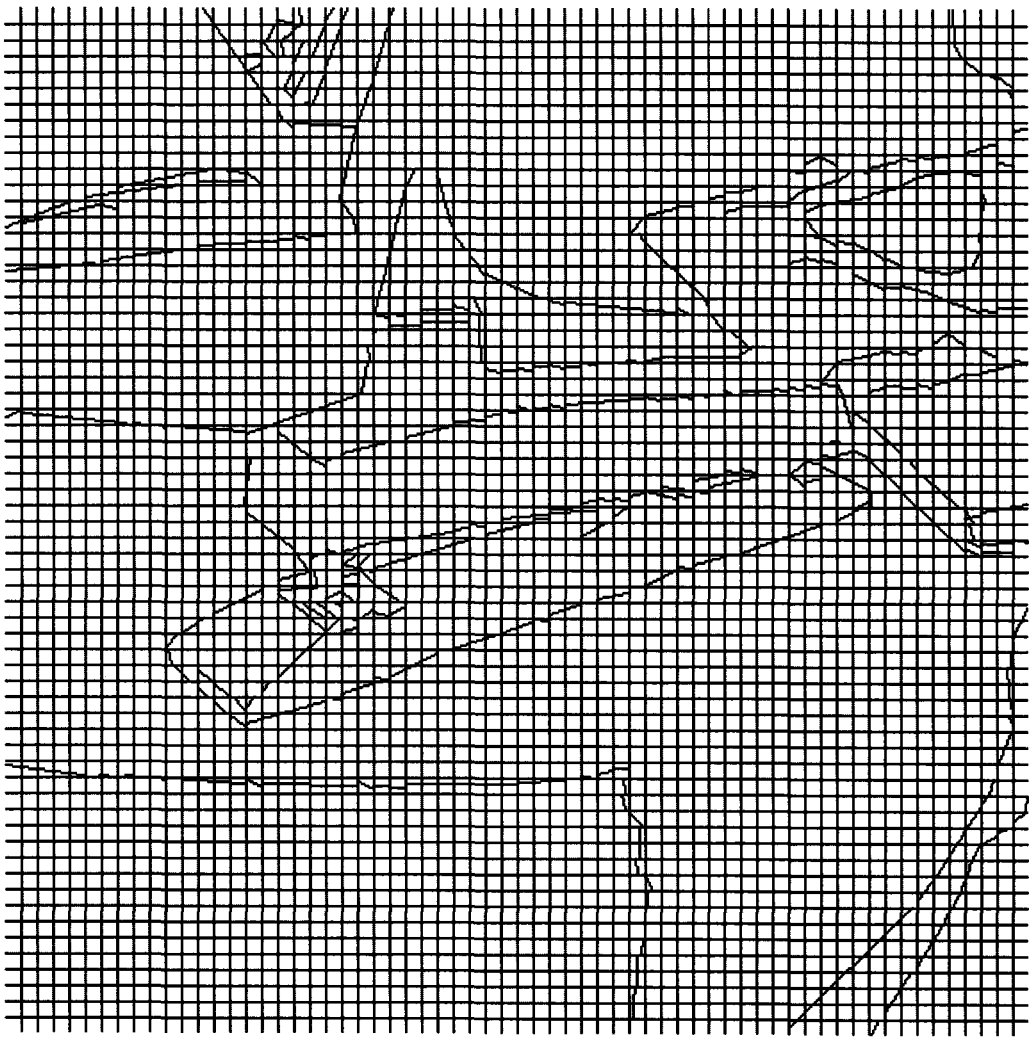
4.2 Grouping Problems

When performing the SDHT with smaller sizes of M, another problem, aside from larger quad-tree sizes and memory storage requirements occurs. This is the problem of trying to group lines found in the images. This is not a new problem however. Some

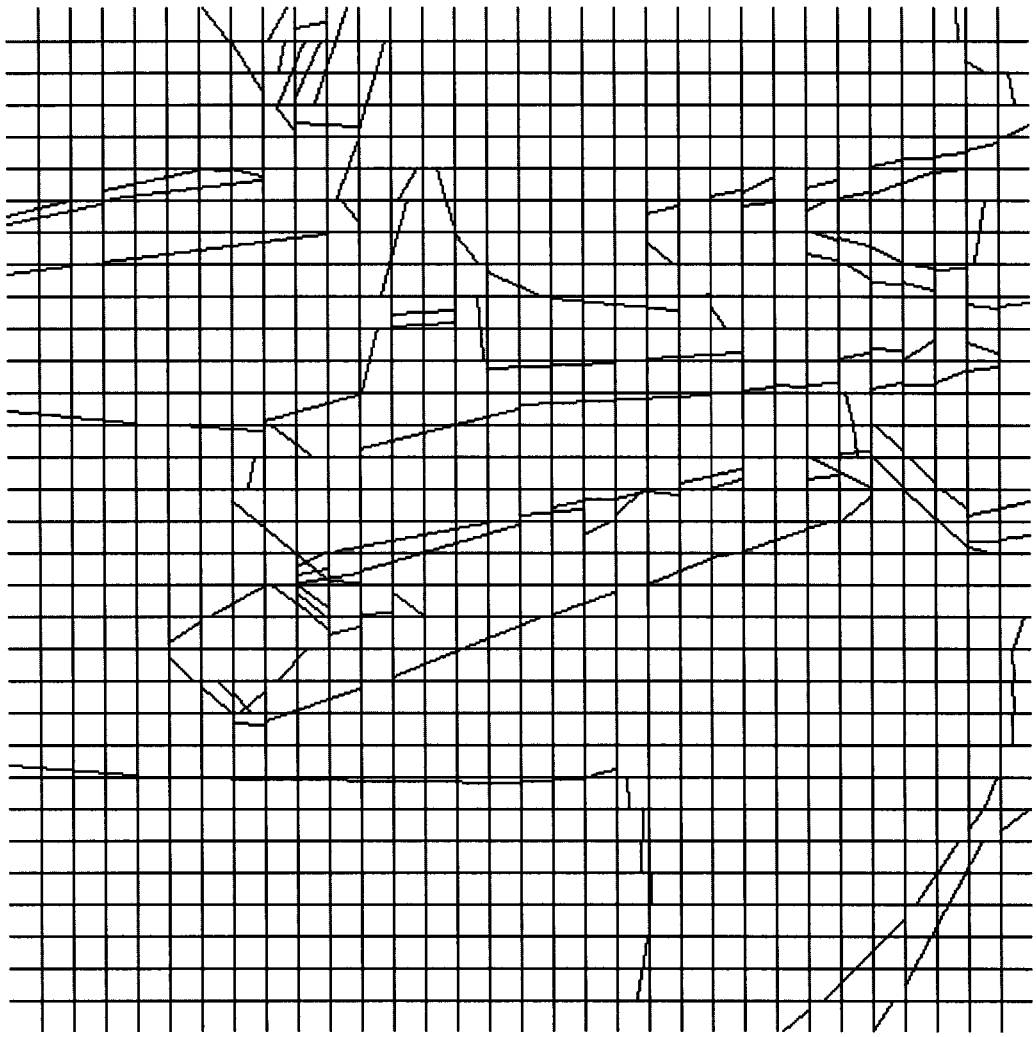
interesting research into the perceptual organization of line segments has been done in [13]. In this thesis we intend to look at this problem in a more natural way from the general framework provided by the SDHT. Although the SDHT reduces the problem of finding co-linear line segments to checking their proximity in parameter space, this can be more difficult than it first appears when sub-image sizes decrease. The smaller a sub-image is the more reduced the spatial resolution and therefore, the less information we have about the orientation of the line segment. Since edge detection can often result in noisy edge images we have the problem of trying to discern if two line segments correspond to the same line when they seem to be a fair distance from each other in parameter space. We find ourselves in the difficult situation of having to choose some sort of maximal distance in parameter space that we are willing to accept as evidence of co-linearity. When the sub-images are smaller, the slightest bit of noise makes a much larger contribution to the overall voting pattern. What we end up with are several short lines, which are at different orientations from each other, but clearly are part of the same line in image space. When the sub-image sizes increase, the noise becomes less of a contributing factor and thus the lines are easier to match up in terms of proximity in parameter space, and thus, co-linearity. It may seem then that the best solution to the problem is to use much larger sub-image blocks. This is only true to a certain degree however, because as larger sub-images are chosen, less information about the locality of the end points of the line segments can be obtained. So there is always this trade-off in results between the sub-image size and the quality of the results. Larger sub-images will give stronger evidence for the co-linearity of line segments while giving less information about their exact positions. Often a legitimate line segment will be picked up on and

extended further than it should due to a lack of more refined knowledge as to its locality. Smaller sub-images will give much weaker evidence for the co-linearity of line segments, and will require a larger distance measure to determine if they are in close proximity in parameter space, but they will give more information about the actual end point locations and length of the line segments. The following Figures, 4.1 (a), (b) and (c) display this trade-off. It can be seen in Figure 4.1a that with a block size of $M = 8$ the SDHT provides the most accuracy about line segment location but many of the longer lines are still fragmented because they could not be grouped properly. A block size of $M = 32$, in (4.1 c) has the reverse problem but $M = 16$ in (4.1 b) works quite well for our intents and purposes. It is also important to note that what we are searching for are strictly straight lines in images. Many images have lines that appear to be straight but have some slight curvature to them, such as in the stapler image below. Therefore, it is natural for some of these lines to be interpreted as different line segments.

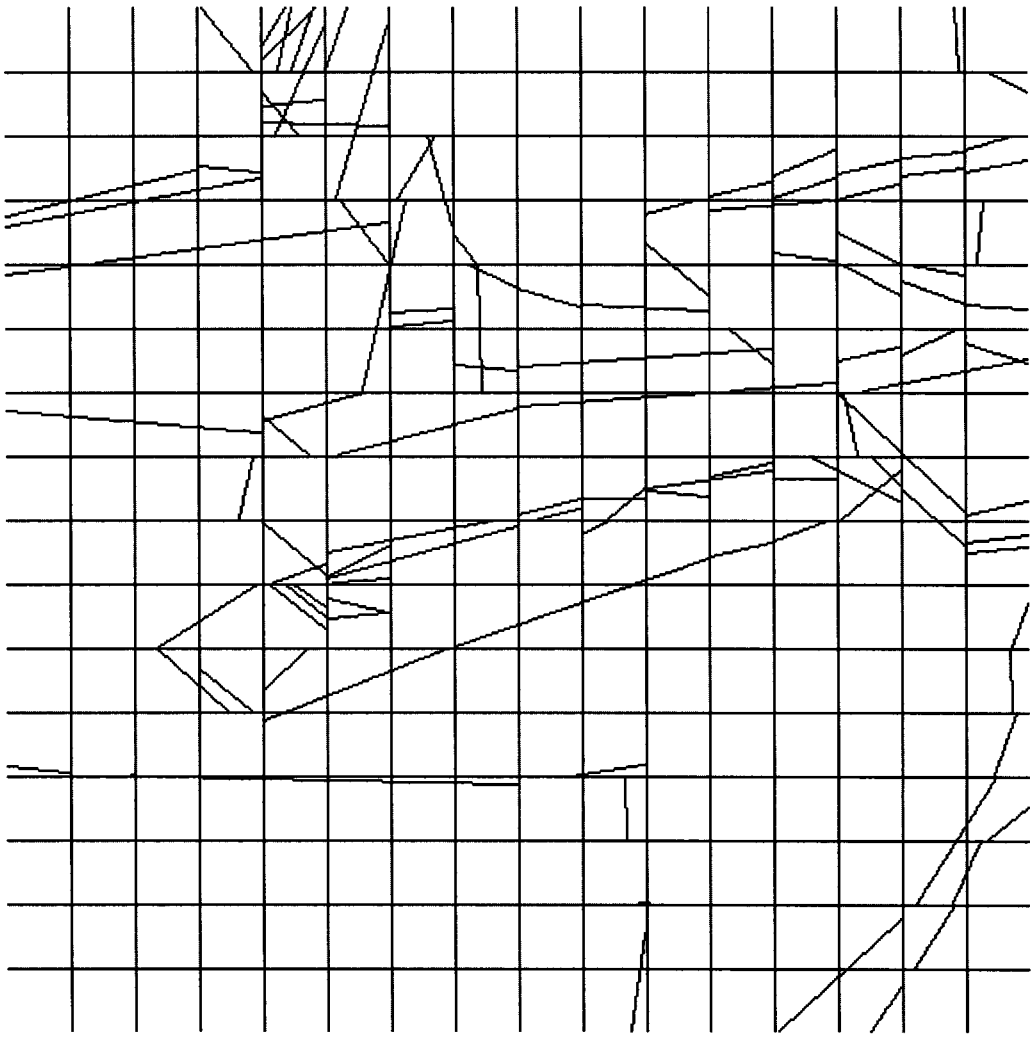
In summary, the spatial resolution of the image has an effect on the results of the line segment detection, as does the parameter space resolution. The larger the block sizes are the stronger the evidence for line segment co-linearity and the less information there is to localize the end points of those lines. It may also be valid to note that if an image has slightly curved lines, such as the one used in Figure 4, then smaller sub-image sizes may not be desirable.



(a)



(b)



(c)

Figure 4.1 (a) SDHT where M=8; (b) SDHT where M=16; (c) SDHT where M=32

Chapter 5

Parameterization

5.1 Importance

In the previous chapters, some key elements of the SDHT were discussed such as the background of the line finding problem in general, as well as spatial decomposition of images and the additive property. These chapters only discussed the transformation process between image space and parameter space. They did not properly address the problem of how to resolve any issues dealing with parameterization. These issues deal with the problems of the HT in general because they are directly related to the implementation of the accumulator arrays used to quantize the parameter space and are not specific to the SDHT. However, due to the problems addressed in Chapter 4, they are even more important with the SDHT than they are with other HT adaptations. Therefore, it is important to re-examine some previously researched ideas to see if any new insights or ideas can be derived which could possibly help for the purposes of the method discussed in this paper.

Parameterization does not just deal with the issues of selecting the optimal equation for the HT, but it must also deal with the issue of making sure that the range constraints for the parameters with regard to the image are valid and that there is minimal redundancy. It must also ensure that the run time is efficient for finding peaks and that the number of false positives in the parameter space is kept to a minimum, among other

things. The following sections will deal with these issues in greater depth and explore some of the pros and cons to various approaches to each problem from the past, while offering suggestions for some new approaches which make more sense with the slightly different paradigm of subdivision being proposed here.

5.2 Choosing Equations

Originally, the HT proposed by Paul Hough used Equation 1.1 to parameterize a line in the image that would be mapped to parameter space. In the previous literature, the problem of this equation being unbounded was resolved by creating two different equations and using two different parameter spaces. The second parameter space would rely on a similar equation to Equation 1.1 where the dependent and independent axes are reversed. As mentioned in the introduction, Equation 1.2 was proposed to solve this problem of Equation 1.1 being unbounded and was renamed the GHT which is most common in literature to this day. However, having two accumulators with Equation 1.1 was still deemed a suitable solution to the problem and merely a petty implementation issue with regard to the overall usefulness of the HT. There is much more to this issue however. When implementing the accumulator arrays the need for two of them and the need to search two of them, creates added problems for run time even though they tend to be somewhat smaller in general. If a technique such as the SDHT is used, this creates a much larger problem because this means that two accumulators must be used for every sub-image in the original image space. The amount of memory can increase very rapidly if this is the case. This has to play a factor when using a hierarchical method. There are other problems though aside from the amount of memory used. Each parameterization is represented slightly differently in parameter space. Since peak values in the accumulator

array represent evidence for lines in the image we must consider the manner of representation of those peaks in parameter space as well. When Equation 1.1 is used, peaks are represented by heavy intersections of lines in the parameter space. When Equation 1.2 is used, however, peaks are represented by heavy intersections of sinusoidal curves in the parameter space. These differences may seem trivial, but they can potentially lead to more problems later on when finding peaks. There are various proposed methods for thresholding out false positives (i.e. peaks corresponding to noise) in accumulator arrays and these methods can become hindered by a poorly constructed accumulator.

5.3 Peak Finding and Thresholding

A peak, by the most basic definition, is any houghel in the accumulator array which is larger in value than all other houghels adjacent to it in the accumulator. This simple definition, while mathematically correct, still requires some modifications because of the false positives that are so prevalent in accumulator arrays. When searching for peaks in an accumulator array the problem of thresholding always becomes an issue. Just as there was a problem with the derivative threshold in edge detection, as described in Section 1.1.1, there are trade-offs that must be accepted with the GHT and any extension of it. For some images and some situations this lack of accuracy would be acceptable, however, for other applications it is not. Therefore, several researchers in the past have looked into this problem in an attempt to find a reasonable solution. One common method involves the search for maximal peaks in accumulator arrays. The idea is as follows. The accumulation array is formed. Then, the houghel with the largest value in the accumulator is found. Once this value is found it is compared with at least some

minimal threshold to see if it is at least worth considering as a line in the image. If this is the case then all feature points that placed a vote for that houghel are removed from the list and the accumulator is cleared before repeating the process again. The pseudo-code can be seen in Figure 5.1.

```
Begin HT(Accumulator, FeaturePointList, PeakList)
     Do
          Clear(Accumulator)
          Accumulate(FeaturePointList, Accumulator)
          Peak = FindMaxPeak(Accumulator)
          If Peak ≥ Threshold Then
               PeakList.Add(Peak)
               For Each Point do
                    If Associated(Point, Peak) Then
                         FeaturePointList.Remove(Point)
                    End If
               End For
          End If
     While Peak ≥ Threshold
End HT
```

Figure 5.1 Max peak re-accumulation pseudo-code

This technique produces perfect peaks every time because it completely eliminates the possibility of false detections caused by overlapping features in the image space. Also, the maximal peak in the accumulator array is always guaranteed to be a legitimate peak regardless of the constraints. The only true question is of the significance of it. The field of image processing is full of these questions regarding the expectations of the user, and therefore, this cannot be helped nor should it be considered a serious flaw to the technique. There are some more serious problems to this though. In a circumstance in which run time is of no concern, this may be an interesting technique to examine when looking for perfect results. However, when run time is of great importance this becomes a major issue because the HT, when applied to an image, is generally a global technique.

Globally an image may have many long lines and many feature points along those lines. If the accumulator array must be cleared and voted on again for every possible line then this could be quite time consuming. One of the benefits is that every time a new line is found the feature points associated with it are removed from the list of viable feature points and therefore, do not require further processing in any re-accumulations. Also, it is important to note that the order in which the lines will be found is in the order of longest to shortest, and therefore, the lines with the most feature points are the ones to be culled from the image first. This is another advantage to the technique. These benefits are not enough however to overcome the re-accumulation in complex images where several lines will need to be found. There is another flaw as well. Suppose that there are thick lines in an image. Suppose also that the step size of the parameters in the accumulation array, are set to be for lines that are only one pixel width thick and are one pixel width apart from each other. In this situation the parameter space will have several patches where several neighboring houghels will contain near identical values representing close parallel lines. When the above technique is used it will return the maximum peak in the accumulator array which identifies a line in the image but when the corresponding feature points are removed from the list, only the ones that are directly responsible for that exact line in image space are the ones that will be removed. Therefore, the close parallel lines in continuous image space that are actually part of the same thick line in quantized image space will remain with their feature points in the list to vote in the next iteration of the loop. This means that the same thick line will be found as several thin lines within the thick line and will detract from the overall understanding of the line being extracted from the sub-image. Some thick lines are difficult to avoid in binary edge images due to

inconsistencies in the gradients supporting each line. Some lines are well supported while other lines we also desire to extract are not. Therefore, setting a proper threshold to acquire all desired lines is sometimes not possible. This problem can sometimes be solved with line thinning algorithms.

The question remaining is as to how this technique applies to hierarchical methods of the HT such as the pyramid methods like the one described in [6, 17, 19], as well as the SDHT described in this paper. This technique was used with great success in [17] and managed to get line segments of excellent quality within a reasonable time frame given the technology of the time. The reason for this was due to the nature of the pyramid technique. The technique allowed for much smaller accumulator arrays corresponding to each overlapping sub-image. This means that the time taken to search for maximum peaks, as well as re-accumulate was greatly reduced, and so the rather brute force nature of the above described technique became acceptable, given the new circumstances. The implementation of this technique however, was rather messy and ad-hoc. The accumulators described had houghel elements defined as the following:

```
typedef struct Houghel
{
     unsigned short value;
     PointList pl;
} *LPHoughel;
```

Figure 5.2 Definition of Houghel Structure for linked list.

As can be seen above, each houghel had a linked list of feature points associated with it in order to record each feature point that placed a vote there. The voting sub-procedure performed had to be modified to add the point currently being processed to the point list shown above. This is not a serious modification but is somewhat messy due to

the fact that clearing the accumulator array means emptying a linked list for each houghel. The maintenance and dynamic allocation of so many lists of feature points certainly is not desired. Another slightly different approach would have been to use a typical houghel definition and merely traverse the full list of feature points afterwards to solve the Equation 1.2 where the parameters are equal to the peak houghel found in the accumulator array. Any feature point for which the equation can be solved, must have voted for that houghel, and therefore, should be removed from the list for the next iteration of the algorithm. This approach was justified in [17] however, due to the small size of each accumulator being used to represent the sub-images.

With the SDHT, this approach can be used to acquire optimal results. However, it suffers somewhat of a time penalty, due to the much larger size of the accumulators, as described in Chapter 4. This time penalty is not as bad as it may seem at first glance however because there are several factors to take into consideration which aid in the speed and design of the algorithm. Finding the global maximum value in an accumulator array, even if the accumulator is rather large, is still a relatively fast process, and certainly does not take as long as finding every local peak by scanning some local region for each houghel entry. Therefore, even with the larger accumulator sizes proposed by the SDHT, this is not a significant problem. In addition, as mentioned before we need not maintain a linked list of feature points for each houghel in the accumulator in order to determine which feature points to remove. We may simply solve Equation 1.2 for each feature point to determine which ones contributed to the best line and then remove each feature point for which Equation 1.2 can be satisfied. This, in and of itself may seem to be a rather lengthy and time consuming process but this is not always the case for two reasons.

The first reason is that the sine and cosine functions, which are typically time consuming, may be pre-calculated in look up tables for the values needed. If this is done then at only a slight increase in memory, Equation 1.2 becomes trivial to solve quickly in terms of run time. Only two multiplications and one addition would be required. For a step size in θ of 1°, the sine and cosine tables would each only require an array of floating point values with a size of 180. By doing this, the run time of the accumulation process is improved as well. The second reason is that we need not always traverse the list of feature points, in order to determine which ones, were the contributing feature points to the best line. This is because we are not interested in re-accumulating unless we know that there will be enough feature points left over to possibly form another line in the parameter space. After removing the contributing feature points (i.e. points which satisfy Equation 1.2), we may check the size of the list. This list size will allow us to determine if we wish to re-accumulate. However, before we even go through this process we already have the information needed in order to determine how many feature points will be remaining afterwards. This is because we may simply take the total number of feature points, and subtract from that number, the total number of votes received by the global maximum (i.e. global peak) in the accumulator array. Due to the fact that this number represents how many feature points contributed to it we know how many should be left over afterwards. If this number is not sufficient for forming another significant line then we may simply clear out the list of feature points and consider the processing for this sub-image to be finished.

When sub-dividing the edge image like this and extracting the feature points from the sub-images first, we make it possible to cull some feature points from the length of a

more global line which are not of particular significance and are not worth accumulating. This makes the technique faster this way, than it would be when used globally. Random noise that falls on the line in other areas of the image will be removed and not considered as they would use the HT as a global technique. Also, it is important to remember that although this iterative re-accumulating technique must run until there are no more lines for each sub-image it is rare for any sub-image to have more than two lines intersecting it. Therefore, the accumulation almost never happens more than once anyway and due to the advantages explained above, the amount of processing required to determine if a re-accumulation is necessary is trivial and can be done in $O(1)$ time. This is especially true once the sub-images reach a size of $M = 32$ or lower.

There is another rather simple yet interesting method though that was briefly mentioned in Chapter 1 that must be revisited. This method when combined with the SDHT could yield an improvement to the method. In [20], a modified houghel definition was used as was described in the introduction. This method, while being a global adaptation to the HT, managed to use a modified accumulator in order to achieve local information about lines. The premise was that the absolute ranges of the lines were recorded in the houghels during the voting process so the end points could be verified later without having to verify their entire geometric path. This is an interesting idea that could perhaps be combined with the SDHT because it would allow for a 25% decrease in the amount of memory required for the final decomposition, while allowing some very accurate results when finding the end points of the lines. The reason for this is that in the SDHT, each houghel contains a 2-byte integer to record the total number of votes accumulated. The memory required for a quad-tree implementation with a depth of d is

always equal to the memory required at depth d-1 multiplied by 4. Therefore the memory required would be $(4d - 4)$. If we were to combine the idea explained in [20] however with the SDHT, then every houghel would have two more 2-byte integers in order to record the ranges line segments. This means that the size of each houghel is multiplied by 3. Therefore, given that this technique finds the absolute end points of the lines, we could combine it with the SDHT to gain one more level of depth in the tree and so the cost in memory would be $(3d - 3)$ instead. Due to the fact that the image has already been decomposed, we can trust with relative confidence that the end points recorded in the line segments are the actual end points and not the result of noise in the image. This is particularly true as the sub-image sizes get smaller. The added run time to accumulate would be within reason and the technique would benefit from the trade-off.

5.4 Range Constraints

The implementation of the HT has been described rather inconsistently in the literature and does not always require the same constraints depending on the circumstances surrounding the application in question. Therefore, it is crucial that a proper explanation is given for the various constraints and the reasons why they were chosen for the examples shown in this paper. Due to the fact that not all lines on the Cartesian plane pass through the image, suitable values must be chosen for the minimum and maximum values of ρ and θ. Upon inspection, it becomes apparent that if the image is N^2 then the constraints must be as shown in Equations 5.1 and 5.2. The sign of ρ can sometimes be determined by the range of θ, as can be seen in Equation 5.3.

$$-\frac{\pi}{4} \leq \theta \leq \frac{3\pi}{4} \qquad (5.1)$$

$$-\frac{N}{\sqrt{2}} \le \rho \le N\sqrt{2} \qquad (5.2)$$

$$\left(0 \le \theta \le \frac{\pi}{2}\right) \Rightarrow \rho \ge 0 \qquad (5.3)$$

The constraint on ρ can be shown in the construction of two isosceles triangles such that the base of one is formed by the upper border of the image and the other by the left-most image border. Each triangle has two 45° angles at the base and one 90° angle opposite the base. This construction illustrates the full range of lines which can possibly intersect with the image. Figure 5.3 illustrates this construction. Although, in theory all gradient lengths ρ should be positive, Equation 5.2 shows that this is not always the case. Therefore, it is necessary to account for all possible negative gradient lengths which could be represented in the mapping process. Given that the image is N^2, the bases of the triangles constructed must also be N. The other triangle segments, excluding the bases, must all be congruent to each other, and have a length of $N/\sqrt{2}$.

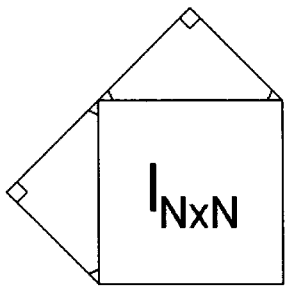


Figure 5.3 Isosceles Triangles demonstrating ρ range constraints.

In [3], Duda and Hart proposed the ranges detailed in Equations 5.4 and 5.5. This would seem acceptable at first glance, however, as was discussed above, there is still a great deal of redundancy with the range chosen for ρ. In [8], the ranges are chosen as shown in Equations 5.4 and 5.6. In addition, [8] asserts Equation 5.7. The above

construction shows that Equation 5.6, although being an improvement to Equation 5.5, still displays some waste when compared with Equation 5.2.

$$0 \le \theta \le \pi \qquad (5.4)$$

$$-N\sqrt{2} \le \rho \le N\sqrt{2} \qquad (5.5)$$

$$-N \le \rho \le N\sqrt{2} \qquad (5.6)$$

$$\frac{\pi}{2} < \theta < \pi \Rightarrow \rho < 0 \qquad (5.7)$$

There is another approach to consider when looking at range constraints though. It is clear to see that no line could ever possibly have a gradient angle θ (as measured from the x axis), between π and $3\pi/2$ or $-\pi/2$. So obviously, it is not necessary to solve Equation 1.2 for any angle in this range. Clearly, only 3/4 of a circle, need to be considered in terms of the θ range. We know however, that the often value of ρ will be negative, and therefore, we must consider the representation of negative ρ values as Duda and Hart did. If, however, we wish to represent all lengths as positive, this can be done by using Equation 5.8 as the range for θ, Equation 5.9 as the range for ρ, and solving Equation 1.2 differently. Basically, only π radians need to be traversed when accumulating but the accumulator itself must have entries to store all the quantized values of θ for Equation 1.2. If a ρ value is positive then it stays as it is. If ρ is negative however, then we simply change the sign to get ρ' and add π to θ to get θ'. The pseudo-code for this can be seen below in Figure 5.4. One thing to consider with this approach is that the accumulator must still consider the full range of Equation 5.8, so the accumulator size must be larger to accommodate it. As well, there is a slight change in the run time of the

accumulation procedure because we must use an "if" statement every time for modification of the (θ, ρ) pair when necessary.

$$\frac{-\pi}{2} \leq \theta \leq \pi \qquad (5.8)$$

$$0 \leq \rho \leq N\sqrt{2} \qquad (5.9)$$

```
Begin HoughTransform(Accumulator)
     For Every Feature Point (x, y)
          For (θ = 0; θ ≤ π; θ += dθ)
               ρ = y*sin(θ) + x*cos(θ)
               If ρ < 0 Then
                    ρ = -ρ
                    θ += π
               End If
               Quantize(ρ, θ)
               ++Accumulator[ρ, θ]
          End For
     End For
End HoughTransform
```

Figure 5.4 HT Pseudo-code using Equations 5.8 and 5.9 as range constraints

Chapter 6

An Application Example

6.1 Introduction to AR

Augmented Reality (AR) which is similar to Virtual Reality (VR) is a relatively new area of research in the field of computer science. Unlike VR which can be defined as the creation of virtual objects in virtual environments, AR can be defined as the creation of virtual objects in real environments. The goal of AR is to integrate those virtual objects into the real environment in real time. AR has several practical purposes. One of which, directly relates to the contribution of this thesis. AR has the ability to enhance a user's perception of the world. It provides information about that world that is not readily available or immediately detectable by their senses. This means that it can be used to amplify their knowledge of a given situation so that they can make more informed decisions. This basic property has allowed for many successful applications such as textual annotations used to give instructions when repairing various devices. It can be used in several areas such as construction and architecture, path planning, scientific visualization, mechanics, entertainment, military devices, Global Positioning Systems (GPS) and even in medicine. The most beneficial application would be medicine, however, because of tracking problems this is only used in training situations due to the degree of error involved and the high degree of accuracy required in the medical field. This problem with tracking is due to the fact that in order for a display device to display a

virtual object overlaid in a real world environment it must know the position and orientation of the user's eyes. This way, it can correctly decide the camera coordinates of the system in order to transform the object properly. Often times though, this information is difficult to obtain and takes more time than is feasible for a real time application. Later, in Section 6.3 however, we shall explain how the SDHT can potentially be beneficial with solving this tracking problem in the field of AR.

6.2 Available AR Technology

There are three different implementation methods related to AR. These methods are listed here as follows:

1. Optical Combining Methods.
2. Video Blending Methods.
3. Monitor Based Methods.

Both of the first two implementations in the list involve the use of a Head Mounted Display (HMD) which is a visor that projects what the user sees into the user's eyes. These displays are different in that they each operate under separate paradigms and each allow for their own set of advantages and disadvantages given various situations. The optical combining methods use optical combiners located in the HMD to reflect light directly into the user's eyes in order to create the illusion of virtual objects in the scene. The real world can still be seen through the lens, however, but at a slightly reduced light. This is because the lens must block some light from the real world in order not to drown out the light from the optical combiners. This is one of the drawbacks to this implementation. However, some HMD devices are sophisticated enough to only block lights from the real world that are at the same wave lengths used by the optical

combiners. In order to perform tracking with this method, a sensor for detecting changes in 3D position and orientation are required.

The video blending method involves the use of a different kind of HMD. With this method, the real world is entirely occluded, similarly to VR. There is a video recording device placed on the HMD which records the scene in front of the user and the HMD displays it. When the image is displayed computer graphics techniques are used to render virtual objects in the scene. The monitor based methods work in much the same way except that a video camera is used to film a scene and the position and orientation of the camera are used to determine the camera mapping matrix for the virtual objects. The recording made by the camera is played out on a monitor, which is viewed by the user. In this sense, the monitor based methods are almost similar to many movies that combine real scenes with computer graphics with the difference of course being that AR is done in real time, rather than being pre-rendered.

One key problem with any method of AR is the actual blending of virtual objects due to the fact that there is not sufficient 3D information if any, for the real world environmental scene. We only know the distance at each pixel for the virtual objects in the scene. So suppose a virtual chair is located at an (x, y, z) position behind a real dinner table. This causes a problem for many AR systems because when the chair is rendered it will not seem as though it is behind the dinner table. This is usually solved with various sensors capable of detecting the distances of real world objects in front of the user (i.e. camera position). This topic, while interesting, is not particularly related to the research in this thesis, and therefore, shall not be discussed in greater detail. In addition, the optical blending combiner methods are also not of particular value with regard to the

SDHT, and shall not be discussed further in favor of the video blending methods. For an excellent survey of AR and the problems associated with it, see [1].

6.3 Video Blending and Robot Vision Connection

As mentioned before, video blending methods allow the system to acquire an image of the real world in front of the camera at any given moment. This added information with regard to the scene is of particular interest to us because it allows us to perform computer and robot vision techniques in order to gain a greater understanding of the situation. The understanding in which we are looking for in particular is that of the position and orientation of the view point of the user. With this information, we are then capable of discerning where the virtual object should be displayed, assuming that it is even within view of the camera at all. Depending on the circumstances and the particular application involved, there may be many different ways in which this information will be acquired. In some cases, fiducials are used in order to track objects that we wish to enhance or annotate in some meaningful way. This eases the tracking problem somewhat because it allows for the recognition of the object in the scene. A typical fiducial might be simply a circle or a square of a predefined color. Locating these kinds of markers in an image is certainly not something of particular difficulty for modern day image processing methods. It is also not a particularly time consuming process either. Using fiducials however may seem to be somewhat unnatural, as it requires us to specifically prepare the environment that we are working in, in order to assist with AR.

As useful as fiducials can be, they always involve the physical modification of the environment in order to acquire assistance in the image processing problem. This is not particularly difficult but it would certainly be of greater interest to us if we could create a

new method for determining position and orientation of the user's eyes based on the natural environment. If there are certain primary pieces of information that already exist naturally in the environment then we wish to make use of them in a meaningful way. One such common feature would be the corner points found in rooms indoors. Typically, most rooms have walls which are orthogonal to each other and a ceiling which is orthogonal to the walls. Only in rare situations with more complex architecture is this not the case. Therefore, we may take advantage of this previously known information in order to derive information about the environment. Whenever a corner exists in a room, it always maps to a 'Y' crossing in the 2D domain (Figure 6a). This 'Y' crossing, if found in the image, could be used to determine the information necessary for AR. The process is as follows:

1. Acquire the lines from the image which correspond with the 'Y' crossing.
2. Determine the end points of the lines to form the 'Y' crossing.
3. Find the angles between the lines in the 'Y' crossing.
4. Map the 'Y' crossing to the 3D domain to determine the distance and orientation of the viewpoint from the HMD.

These steps may seem rather simple but there are some problems that arise when we attempt to do this, such as in step 2. This is where the SDHT is of value. We are interested in knowing if the 'Y' crossing is an orthogonal intersection of two walls and a ceiling, or of two walls and a floor. This is because either intersection could potentially be used to solve this computer vision problem in the field of AR. We are also interested in finding the lines and their respective end points rather quickly because in AR this is of high importance. The user does not want to wait long for the computer to confirm the

position and orientation. There are several problems that are encountered by using a standard HT for this problem, or for that matter, any other form of spatial decomposition related to the HT. With the standard HT there is the problem of having to find all of the lines in the image and then compare them to each other later in order to find a common intersection between 3 lines. This tends to be a difficult problem because nothing is known about the locations of the lines other than their global end points which are found by intersecting them with the boundaries of the image. The legitimate intersection can be calculated between them. However, due to the fact that the 'Y' crossing can be right side up or up side down, we don't know which of the global end points to replace with the intersection point for each line segment.

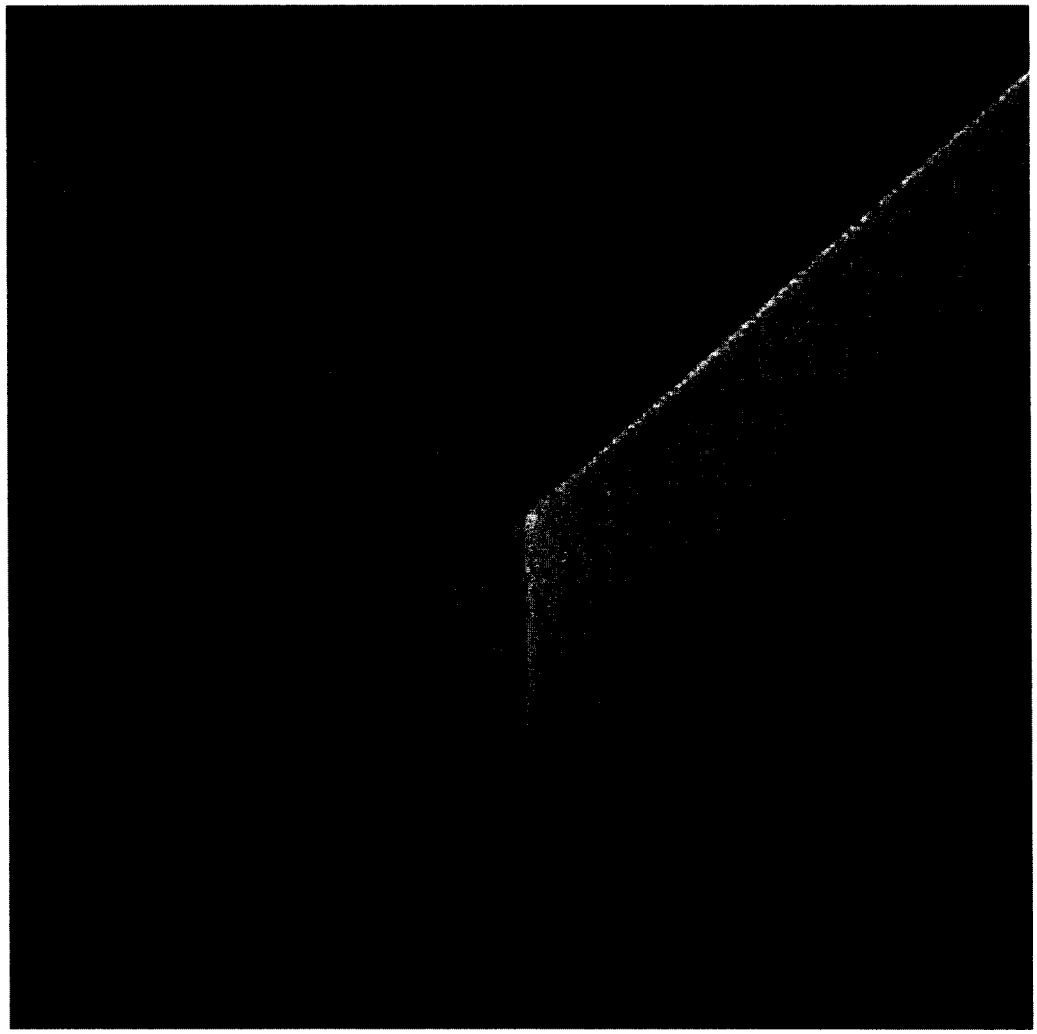
Using a spatial decomposition technique will give us this information, which is a much simplified case of the previous images provided earlier in this paper. The reason for this is that we have a-priori knowledge of the object we are looking at in the first place. Due to this knowledge, we may simply subdivide the image into four equal sections, resembling the decomposition that would result if a quad-tree with a depth of 1 was used in the previous discussions. The process is now much easier to resolve because due to the nature of the scene, there really should only be one true line in each sub-image at most. If the maximum peak value is found to be of significance in each of the accumulators then the proper line has been found in general. So the process is greatly reduced. The end points can be found by simply finding both global end points for each of the 3 lines. One of these end points for each one will be legitimate, while the other one will be a global extension of the line segment. The invalid global end point for each line segment merely needs to be replaced by the intersection of the 3 lines. Due to these simplifications in the

problem domain, it really makes no sense to use a pyramid or quad-tree type structure to perform this, because that kind of refined information is not necessary and would take far too long for an AR application. In addition, it is also inappropriate to spatially decompose the image using a local origin for each line segment. This is because once all of the lines have been acquired, they will all be in different coordinate systems, and therefore, finding their intersection will suddenly be much more difficult. This is why a simplified version of the SDHT is the most appropriate solution for the problem.

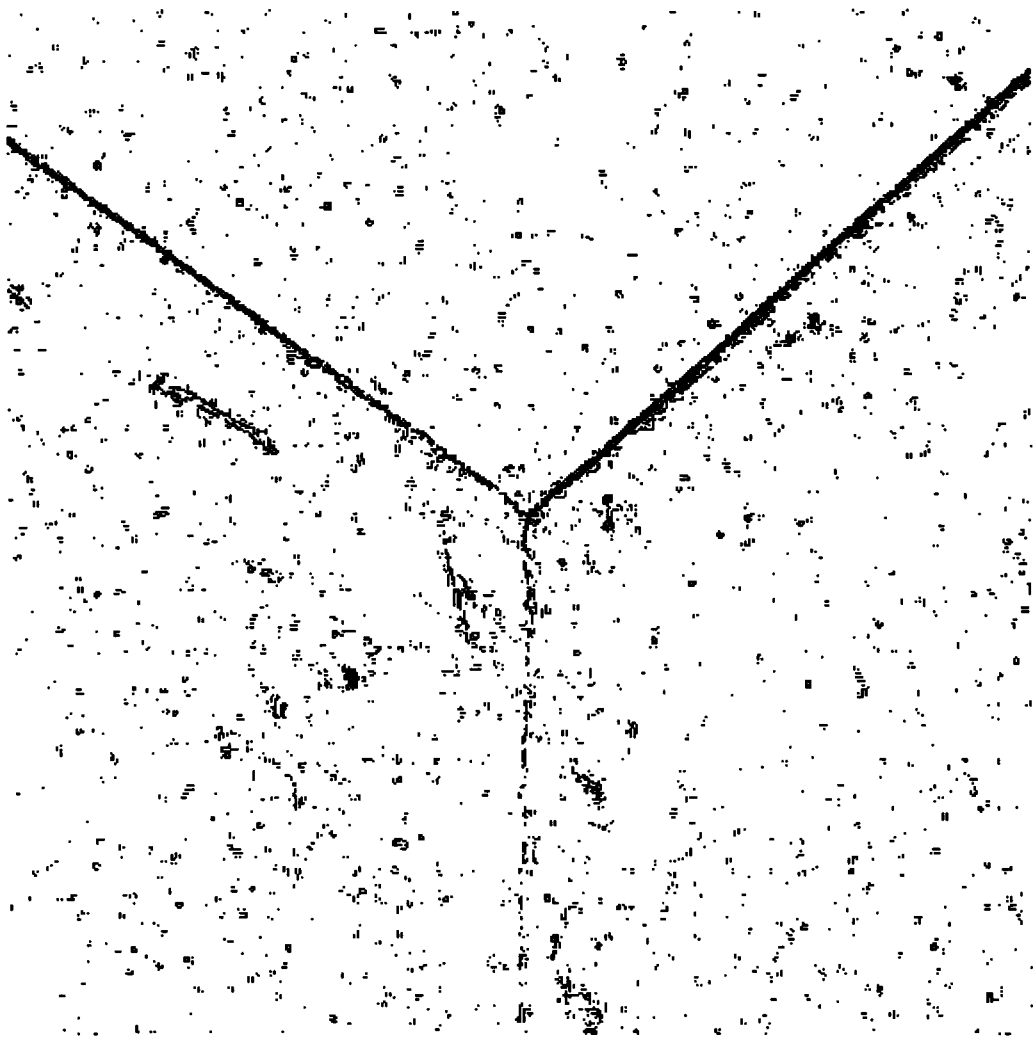
The process simply involves the accumulation of all four accumulator arrays defined from the global origin for each sub-image. Then, the maximum peak corresponding to the top two sub-images are found, in an attempt to search for a right side up 'Y' crossing. If this is done properly, then the bottom two accumulators can be added together by Equation 3.4 to give us the accumulator corresponding to the entire bottom half of the image. Once this is done, the best peak value is found in that accumulator as well and is assumed to be the line segment forming the bottom portion of the 'Y'. A common intersection must then be found between all 3 lines, within a pre-specified degree of accuracy and within a certain proximity to the center of the image. If this can be done, then the spatial information we have already obtained may be used to determine which of the global end points to replace with the intersection point between the lines. If a significant peak value could not be found in each of the accumulator arrays corresponding to the top half of the image, then the reverse process would take place in order to find an upside down 'Y' crossing.

In Figure 6a, there is the problem of the existence of several spider webs in the corner that was used for the image. As well, the limited shading information also makes it

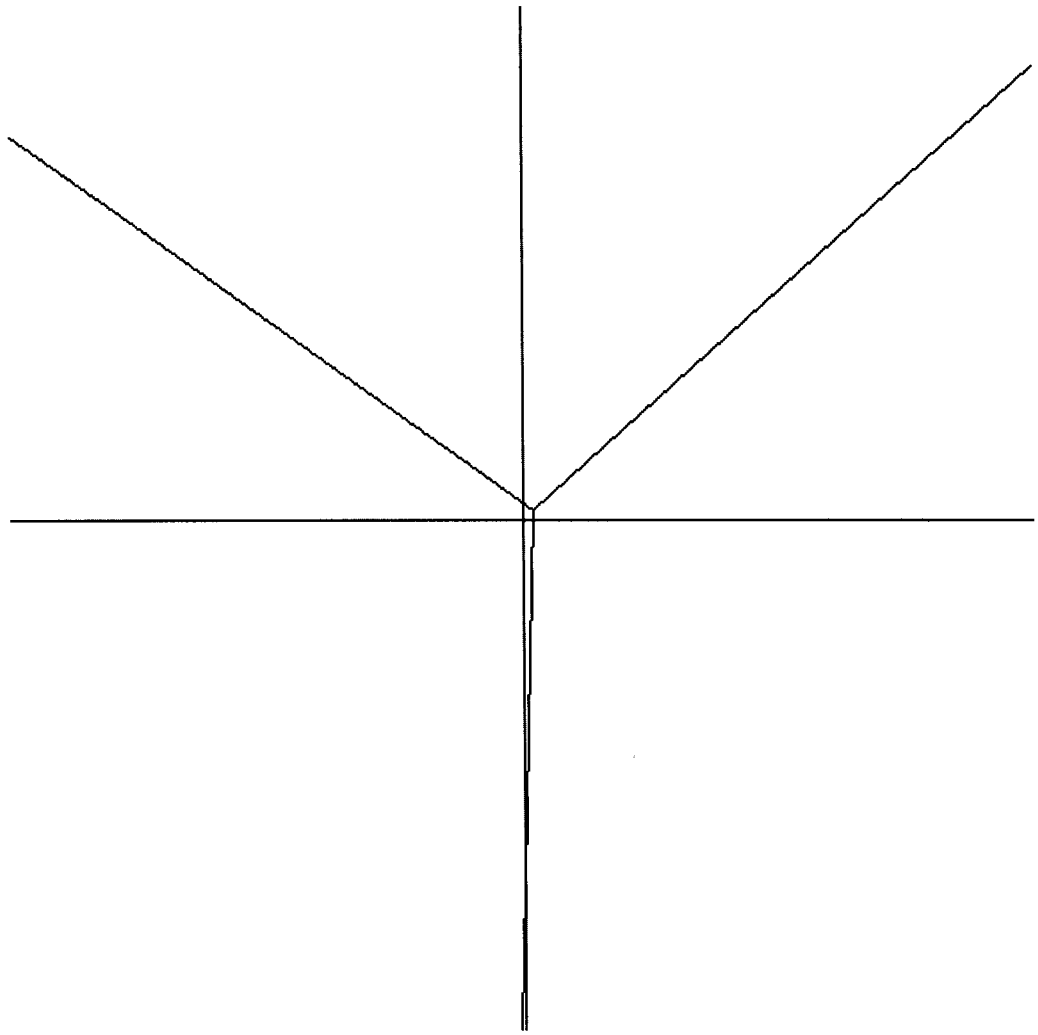
difficult to edge detect properly. This relates to the discussion on edge detection from Chapter 1. In Figure 6b, we can clearly see the effects of these problems. The spider webs are detected by the edge detection algorithm and show up in the binary edge image. They add additional noise that could potentially lead to false positives in other techniques. Also, the bottom line in the 'Y' crossing was difficult to pick up on and can only faintly be seen. This difficulty arises from the higher first derivative threshold required in an attempt to eliminate noise caused by the spider webs. The SDHT however, when used in this particular application, only needs to find the best peak for each accumulator, and therefore, these false detection problems are solved easily and quickly. Even in difficult situations where there is increased noise in the image, these problems can easily be resolved by finding the best peaks possible and ignoring the rest. A common example of this would be a stipple ceiling which tends to add a great deal of noise to a binary edge image. As can be seen in Figure 6c, all 3 lines were found correctly by the algorithm and therefore could be used later in further processing for AR. Only the SDHT and how it relates to this problem was actually implemented and researched however. Further steps in this computer vision process are not part of this research.



(a)



(b)



(c)

Figure 6 SDHT Applied to Indoor Corner Image for Use in AR: (a) Original Image; (b) Binary Edge Image; (c) SDHT Results

Chapter 7

Conclusion

7.1 Thesis Summary

In this thesis we have discussed the general problem of finding lines in images and several approaches to the problem devised in recent years. In particular we have discussed the concept of using hierarchies and spatial decomposition of binary edge images to determine the locality of their end points. We have devised a novel new approach to this spatial decomposition which we refer to as the SDHT. We discussed both the benefits and the drawbacks to the use by the SDHT of a global origin with each accumulator corresponding to a sub-image. The additive property which allows for the implicit storage and efficient acquisition of accumulators to parent sub-images, gives us a new paradigm for the HT and other transforms as well such as the Radon Transform.

The SDHT has the drawback of using larger accumulator arrays but with most modern computers this is not an overly significant problem as it was in the 1980's when this research was more prevalent. The use of these larger accumulators however, allows for the reduction of the line co-linearity problem to a mere proximity problem in parameter space. As well, the ability to directly apply Cramer's Rule to these lines to acquire their intersection points for various applications such as AR has also been achieved. The SDHT serves as an appropriate and efficient choice for the discovery of 'Y' crossings in images which would require more complex, and possibly much slower

techniques otherwise. The iterative modification of the GHT when applied to the SDHT is rather efficient due to the fact that there are very few lines in small sub images and usually only one line needs to be found. The use of Equation 1.2 to determine which points to remove from the feature point list allows us to avoid the problems of maintaining a linked list of points at every node in the accumulator array. The a-priori knowledge of how many feature points will be in the point list (after the points contributing to the previous peak value have been removed), allows us to avoid this process altogether and bypass any further processing with this accumulator if we know that there are not enough significant feature points left.

We have discussed the several approaches taken in the past to choosing the gradient angle and gradient length constraints. We have also introduced a novel new approach that reduces redundancy in the accumulator arrays, particularly along the quantized gradient length. We have proven this choice of range constraints to be more efficient and to have no redundancy.

This thesis has also shown the relationship between the image sizes and the sub image sizes chosen when applying the quad-tree method to the SDHT. We have shown through examples that when the sub-images size increases, there is more co-linearity information present but less spatial location information. We have also shown that when the sub-images decrease in size, there is more spatial location information but there is less co-linearity information, and therefore, proximity constraints in the parameter spaces used to derive co-linearity must be relaxed to achieve adequate results.

7.2 Future Research

There are several different variations to the HT, many of which do not involve hierarchies such as quad-trees or pyramids at all. Many do not even involve any spatial decomposition of the edge images. This paper only discussed the use of the SDHT on the GHT. In the future perhaps, due to the orthogonal nature of the SDHT to all other non-hierarchical approaches, it may be of value to research how other variations of the HT could improve the results of the SDHT. In addition, there may be better ways for the decomposition of the binary edge image. In this thesis we made the suggestion of a quad-tree for the general case. However, as long as Equation 3.4 is satisfied, and the sub-images are disjoint from each other, then there is really no need for the sub-images to all be rectangular. They may be of arbitrary shapes and sizes, depending on the application involved. One particular field of research, known as Attributed Vector Quantization [21], could play an interesting role in choosing more appropriate sub-images in future research.

In this thesis, we have discussed the application of the SDHT to the field of AR and to the specific problem of finding 'Y' crossings in binary edge images. In the future however, it may be of value to find some other geometric patterns commonly found in indoor or outdoor scenes which the SDHT could be specially adapted to. As well, an entire process in AR for determining the position and orientation of the view point for the user was discussed but only the first portion of the process was discussed, as it applied to the SDHT. Future research may be of value to solving the problems associated with the later stages of the process.

References

[1] R.T. Azuma, "A Survey of Augmented Reality", In Presence: Teleoperators and Virtual Environments, vol. 6, pp. 355-385, 1997.

[2] J. Canny, "A computational approach to edge detection", IEEE Trans. Pattern Anal. Mach Intell. PAMI-8, No. 6, pp. 679–698, 1986.

[3] R.O. Duda and P.E. Hart, "Use of the Hough Transform to detect lines and curves in pictures", Commun. ACM, vol. 15, pp. 11-15, 1972.

[4] M. Heath, S. Sarkar, T. Sanocki, and K.W. Bowyer. "A robust visual method for assessing the relative performance of edge detection algorithms", IEEE Transactions on Pattern Analysis and Machine Intelligence 19 (12), pp. 1338-1359, December 1997.

[5] J. Heather and X.D. Yang, "Spatial Decomposition of the Hough Transform", Proceedings of the 2nd Canadian Conference on Computer and Robot Vision, pp. 476-482, 2005.

[6] T.H. Hong et al., "Using Pyramids to Detect Good Continuation", IEEE Trans. Systems Man Cybern. SCM-13, No. 4, pp. 631-635, 1983.

[7] P.V.C. Hough, "Method and Means for Recognizing Complex Patterns", U.S. Patent 3,069,654, 1962.

[8] L. Jin and L. Yang, "Parallel Solution of Hough Transform and Convolution Problems – A Novel Multimodal Approach", Proceedings of the 1992 ACM/SIGAPP symposium on applied computing: technological challenges of the 1990's, pp. 775–781, 1992.

[9] A.L. Kesidis and N. Papamarkos, "A Window-Based Inverse Hough Transform", The Journal of the Pattern Recognition Society, vol. 33, pp. 1105–1117, 2000.

[10] A.L. Kesidis and N. Papamarkos, "On the Inverse Hough Transform", IEEE Trans. Pattern Anal, vol. 21, No. 12, 1999.

[11] Y. Kunii and H. Chikatsu, "Efficient Line Extraction for Digital Archives of Cultural Heritage Using Optical Flow and Trifocal Tensor", 20th ISPRS Congress (Geo-Imagery Bridging Continents), vol. 35, section B5 (5th Commission), p. 914, 2004.

[12] H. Li, M.A. Lavin and R.J. Le Master, "Fast Hough Transform: A Hierarchical Approach", Computer Vision, Graphics, And Image Processing, vol. 36, pp. 139–161, 1986.

[13] D.G. Lowe. "Perceptual Organization and Visual Recognition", Kluwer Academic Publishers, 1985.

[14] C.F. Olson, "Decomposition of the Hough Transform: Curve Detection with Efficient Error Propogation", Proc of the European Conference on Computer Vision, pp. 263–272, 1996.

[15] C.F. Olson, "Improved Curve Detection Through Decomposition of the Hough Transform", Technical report 95-1516, Department of Computer Science, Cornell University, 1995.

[16] A.P.D. Poz, G.M.D. Vale and I.R.B. Zanin, "Automated Road Segment Extraction by Grouping Road Objects", 20th ISPRS Congress (Geo-Imagery Bridging Continents), vol. 35, section B3 (3rd Commission), 2004.

[17] J. Princen, J. Illingworth and J. Kittler, "A Hierarchical Approach to Line Extraction Based on the Hough Transform", Computer Vision, Graphics, And Image Processing, vol. 52, pp. 57–77, 1990.

[18] P.L. Rosin and G.A.W. West, "Segmentation of edges into lines and arcs", Image and Vision Computing, vol. 7, pp. 109-114, 1989.

[19] M. Schneier, "Two Hierarchical Linear Feature Representations: Edge Pyramids and Edge Quadtrees", Comput. Graphics Image Process. 17, pp. 569-572, 1981.

[20] J. Song, M. Cai, M.R. Lyu, and S. Cai, "A New Approach for Line Recognition in Large-size Images Using Hough Transform", Proceedings of the 16th International Conference on Pattern Recognition (ICPR'02), vol. 1, p. 10033, 2002.

[21] A. Ward, "Attributed Vector Quantization: A New Approach to Pattern Acquisition and Recognition", M.Sc. Thesis, Department of Computer Science, University of Regina, Canada, 2004.

[22] X. Wu, K. Wang and D. Zhang, "A Novel Approach of Palm-line Extraction", 3rd International Conference on Image and Graphics (ICIG'04), vol. 1, pp. 230-233, 2004.

[23] D. Xi and S. Lee, "Reference Line Extraction from Form Documents with Complicated Backgrounds", Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR'03), vol. 1, pp. 1080-1084, 2003.

[24] W. Zorski, B. Foxon, J. Blackledge and M. Turner, "Fingerprint and Iris Identification Method Based on the Hough Transform", http://www.iar.wat.waw.pl/strtekst/biuletyn/biul15/A4.pdf, 2001.