Review of the Hough Transform Method, With an Implementation of the Fast Hough Variant for Line Detection

Danko Antolovic
Department of Computer Science, Indiana University,
and IBM Corporation

Abstract

This report explains the basic principles of the Hough Transform method for detection of geometric shapes, and reviews some of its variants and generalizations. The report further describes an implementation of the Fast Hough variant of the transform, for the detection of lines and circles in the framework of an image-processing application. Considerations of speed and efficiency of the Fast Hough algorithm are discussed in detail, with a view toward an ASIC implementation.

Introduction

The Hough Transform method was introduced, in its most elementary form, by P.V.C. Hough in 1962, in the form of a patent [1]. Its intended application was in particle physics, for detection of lines and arcs in the photographs obtained in cloud chambers. Many elaborations and refinements of this method have been investigated since.

The Hough Transform (HT) falls into the midrange of vision-processing hierarchy. It is applied to images which have already been freed from irrelevant detail (at some given size scale) by some combination of filtering, thresholding and edge detection. The method attributes a logical label (set of parameter values defining a line or quadric) to an object that, until then, existed only as a collection of pixels; therefore it can be viewed as a segmentation procedure.

The idea behind the method is simple: parametric shapes in an image are detected by looking for accumulation points in the parameter space. If a particular shape is present in the image, then the mapping of all of its points into the parameter space must cluster around the parameter values which correspond to that shape.

This approach maps distributed and disjoint elements of the image into a localized accumulation point, which is both a benefit and a drawback. Partially occluded shapes are still detected, on the evidence of their visible parts – for example, all segments of the same circle contribute to the detection of that circle, regardless of the gaps between them. On the other hand, local information inherent in the points of the shape, such as adjacency, is lost - endpoints of circle arcs and line segments must be determined in a subsequent step.

Computational load of the method increases rapidly with the number of parameters which define the detected shape. Lines have two parameters, circles three, and ellipses (circles viewed at an angle) have five. Hough method has been applied to all of these, but the ellipse is probably at its upper limit of practicality. Attempts have also been made to

apply the Hough method to arbitrary shapes. These attempts essentially reduce the method to template matching, canceling the benefits that derive from the analytic, few-parameter description of the shape.

By its definition, the Hough transform has limited perceptual scope. Its greatest strength lies in specialized vision, such as manufacturing quality control, analysis of aerial photographs, and (one would hope) data analysis in particle physics. It has little in common with the general-purpose vision of animals; its role in the vision systems of free-ranging robots would likely be limited to some specialized subsystem.

Line Detection and Line-Point Duality

Line detection is based on simple point/line duality. If a line in the image is defined by points whose coordinates satisfy the equation

$$y = kx + c \qquad (1)$$

then each point can also be seen as defined by the bundle of lines passing through it. A point P in *x,y* space defines, by equation (1) a line p in the *k,c* space; each (*k,c*) point ℓ on the line p defines, in turn, a line L in the bundle of lines passing through P in the *x,y* space.

For all the points P on a line L in *x,y* space, corresponding lines p form a bundle intersecting at a point ℓ in *k,c* space. Coordinates of ℓ are the parameters of line L, therefore this method can be thought of as a transform between spaces *x,y* and *k,c* (see Figure 1).

Now, let L be a line that is present in the image (*x,y* space), traced by some black pixels, possibly spread over the image in small fragments and mixed up with non-linear features. The line must nevertheless have a corresponding point ℓ in the parameter (*k,c*) space.

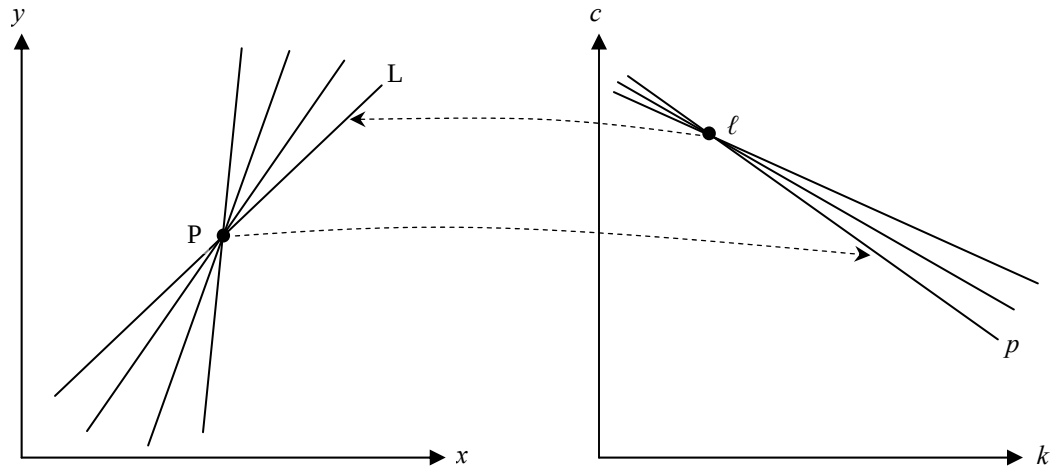

Figure 1. Line-point duality: point P (line bundle) in the image defines line p in the parameter space; point ℓ (line bundle) in the parameter space defines line L in the image.

Let us also divide the parameter space into a raster of suitably fine cells. Each image pixel traces a line p in the parameter space, and the passage of that line through each cell is recorded. The cell corresponding to point ℓ accumulates large numbers of hits, and thus identifies the line L in the original image.

Non-linear features, on the other hand, contribute only to the background distribution of hits. Separation between the two is a matter of choosing some suitable threshold, and it is obvious that linear features with few pixels can be lost in a large volume of non-linear chaff.

Normal-form Parametrization of the Line

Since the equation (1) has infinite slope for the vertical line, Duda and Hart [2] introduced a singularity-free alternative, the line parametrization

$$\rho = x\cos\theta + y\sin\theta \qquad (2)$$

Here ρ is the line's distance from the origin, and θ is the inclination of the normal. Like Equation (1), this formula defines a mapping between the image (x,y) space and the parameter (ρ,θ) space; however, image points define sine/cosine curves in the parameter space, rather than lines. Using ρ and θ as the line parameters removes the slope singularity, but at the cost of computing trigonometric functions. Since these functions only need to be computed (tabulated) for one dimension of the parameter raster, the cost may well be acceptable.

A line in the x,y space is represented by the point of intersection of sine curves, in the characteristic "butterfly" pattern (Figure 2.), and the line detection consists of finding accumulation points of curves. Determining whether the point's transform intersects a (presumably) rectangular cell of the raster is more complicated for curves than for straight lines; see the discussion of the Fast Hough parameter search below.

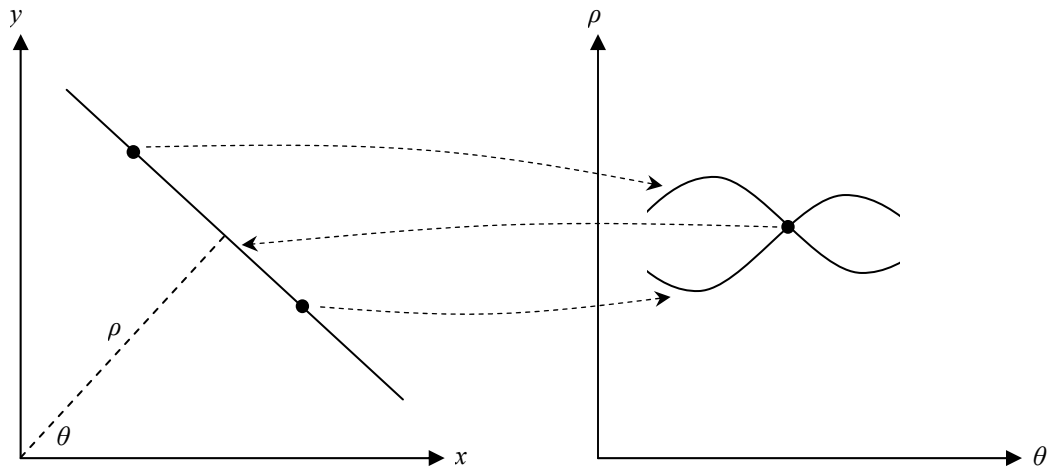

Figure 2. Normal-form parametrization and the butterfly pattern

Detection of Circles and Ellipses

The above method(s) can, in principle, be generalized to parametric curves, with the implication that points in the *x,y* space map into surfaces in multi- (3 to 5) dimensional parameter space. Storage requirements make this approach largely impractical, and Hough method is often supplemented with information about edge direction, in order to reduce the search in the parameter space.

In particular, Illingworth and Kittler [3] proposed using the Sobel operators to obtain edge direction at each image point, and then detect circle centers as the accumulation points at which edge normals intersect. A simple histogram detects the accumulation points in the "radius space."

Calculating the edge direction is typically rather inaccurate and introduces amplified errors in the location of the circle's center. If large convolution masks are used for more accurate edge detection, computational load increases in turn. We have found that using secants between pairs of points works very well for finding circle centers, introducing smaller errors than the local edge direction. For a comparative study of the methods for circle detection, see Yuen et al. [4]

V.F. Leavers [5] has introduced the double transform as an interesting method of reducing the parameter space for circle/ellipse detection. It consists, first, of line-transforming each point on the curved shape, using the normal-form parametrization. These transforms form a family of curves in parameter space, and the envelope of the family represent all tangents to the original shape in the image. A second transform detects flat stretches on the enveloping curve; from these, parameters of the original shape can be derived.

Probabilistic Hough Transform

Due to the correlated and redundant nature of visual information, all vision algorithms are computationally intensive, either in processor time or in storage requirements. HT is no exception, but, as discussed above, it does not use the local, neighboring-pixel information to accumulate evidence for a parametric shape. It is therefore fairly stable under random sampling of image data. We have used random sampling of edge images, and have found that line and circle detection works quite well even with a fairly small subset of image points. For a review of probabilistic methods, see Leavers [6].

Generalizations to Arbitrary Shapes

An early generalization of the Hough method to arbitrary shapes was given by Ballard[7], who proposed describing points on an arbitrary curve as a multi-valued tabulation of radii vs. angles, in a radial coordinate system centered at some point on the curve. Not surprisingly, such tabulation undergoes simple transformation under rotation

around the coordinate origin and isotropic scaling. The remainder of Ballard's method is template matching against the tabulated shape, under the transformations of scaling and rotation. Its limitation is summarized by Leavers: "… a particular shape can only be efficiently detected if one knows the transformation from a canonical form of the shape to the instance of that shape. If the transformation is not known then all plausible transformations must be tried." (Leavers, [5], p.116)

The reason why standard HT works so well for lines, and to a lesser degree for quadrics, is that it exploits analytical properties of these shapes. All generalizations of HT to arbitrary shapes ultimately reduce to template matching, as they must, with a large search space and little promise of computational speed. As an illustration of that principle, Stockman and Agrawala [8] show the connection between HT and template matching for the simplest HT case, the classic line detection.

Literature based on the Hough transform covers a vast area, involving methods of very mixed quality. The classic survey article is by Illingworth and Kittler [9], and other recommended sources are Leavers [5], and a comprehensive textbook on machine vision by E.R. Davies [10].

Implementation of the Fast Hough Transform

In every HT variant, major computational effort goes into scanning the parameter space, in order to determine how many point transforms (lines, curves or surfaces) intersect each cell. One expects to see clusters of intersections, corresponding to prominent image features, and several methods have been proposed to find these features efficiently.

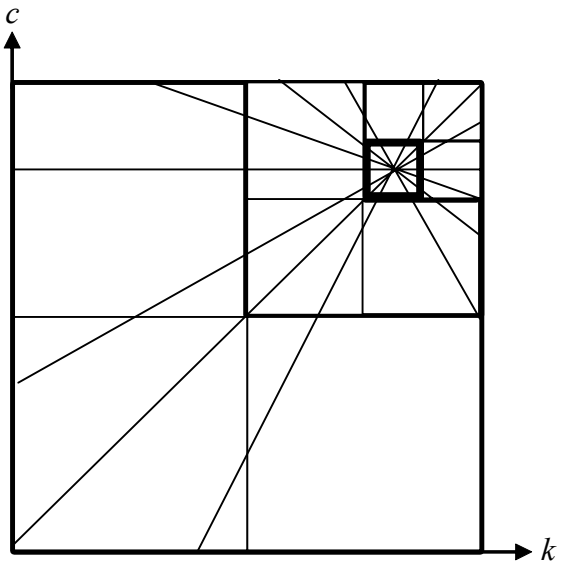

Figure 3. Example of a quadtree search in the FHT (after Li et al. [11]). Entire parameter space is considered first, then its upper right sub-quadrant, etc. Far from growing exponentially, the search converges rapidly on the accumulation point.

The method discussed here was introduced by Li, Lavin and LeMaster [11], under the name “Fast Hough Transform” (FHT). It divides the parameter space into “hypercubes” arranged in a tree structure – in the 2D case, quadrants of the plane are repeatedly divided into sub-quadrants, and arranged into a search quad-tree. Exponential growth of the search tree is easily prevented by pruning it down to promising quadrants only (those with many intersections), and in fact the search converges rapidly (see Figure 3). The infinite-slope problem can be resolved by using an “inverted” (*c,k)* space when $|k| \geq 1$.

The crux of the computation lies in determining whether lines in parameter space (transforms of image points) intersect the quadrants of the search tree. As shown in Figure 6, line $c = Ak + B$ intersects the larger (parent) quadrant, and the section is fully determined by the values of the function $z = Ak - c + B$ at any three corners of the quadrant. The task is to determine efficiently which of the four child quadrants this line intersects; we relegate the discussion of the algorithm to the Appendix.

We have implemented the FHT for lines and circles, as part of an image-processing package [12]. The FHT is accompanied by several auxiliary processing steps, whose purpose is to speed up the parameter-space search. The implementation is fully sequential, and can be represented by this (very simplified) pseudocode:

```
DoHoughTransform() {    // entry point to Hough transform

      Perform thinning on the entire image;          // see [13]
      Divide image into tiles;

      for (loop over tiles) {

            Find connected components within a tile;  // see [14]

            for (loop over connected components) {

                  while (there are unassigned feature points) {

                        Random subsampling of points;
                        FHT for lines;
                        FHT for circles;
                  }
            }
            Find line segments of the found lines;
            Find circle arcs of the circles;
            Clean up very small or irregular features;
      }
}
```

Test Results and Conclusion

Figures 4 and 5 are examples of interpretation of line sketches, performed by the above image-processing program. The thin line drawings are the hand-made pixmap input, and the boldface sketches are the CAD-like output, consisting of parametric elements (line segments and circle arcs).

As expected, HT performs better for larger shapes. Small circles and short lines yield less well-defined accumulation points, simply because fewer pixels contribute to the accumulation. Consequently, small elements are recognized somewhat more ambiguously than the large ones (e.g. details in the upper right corner in Figure 5).

Spatial relations are lost in the transformation and recovered in a subsequent image scan. The consequence is the appearance of phantoms, since anything that lies very close to a recognized line/circle is counted as part of it. For example, the short line segment in the steeple window in Figure 4 lies on the line defining the roof . Even more prominently, an unrelated line is tangential to the bottom part of the circle defining the cupola, giving rise to a small phantom circle arc between the two round windows.

These are straightforward disambiguation problems, which can be resolved by attributing some plausible priorities to the detected parametric shapes. However, they illustrate the distance that has to be traveled on the way from a mathematical method to a working vision system.

The core of the FHT, the hierarchical search in parameter space, is a remarkably simple recursive calculation, consisting of integer arithmetic. Since the Hough Transform inherently treats image pixels as independent from one another, per-pixel operations are parallelizable. In a real-time implementation, each step of the recursive search should consist of parallel image-point transforms and intersection decisions, followed by an accumulation of hits in the search-tree quadrants. A parallel ASIC, combined with microprocessor core(s), is a promising architecture for this algorithm.

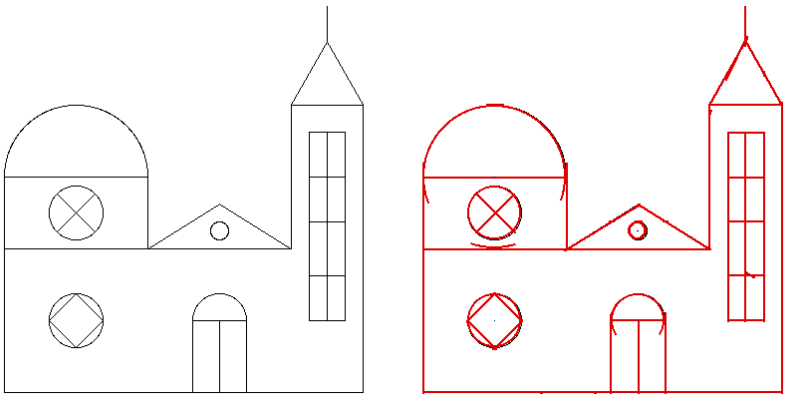

Figure 4. A fanciful building facade

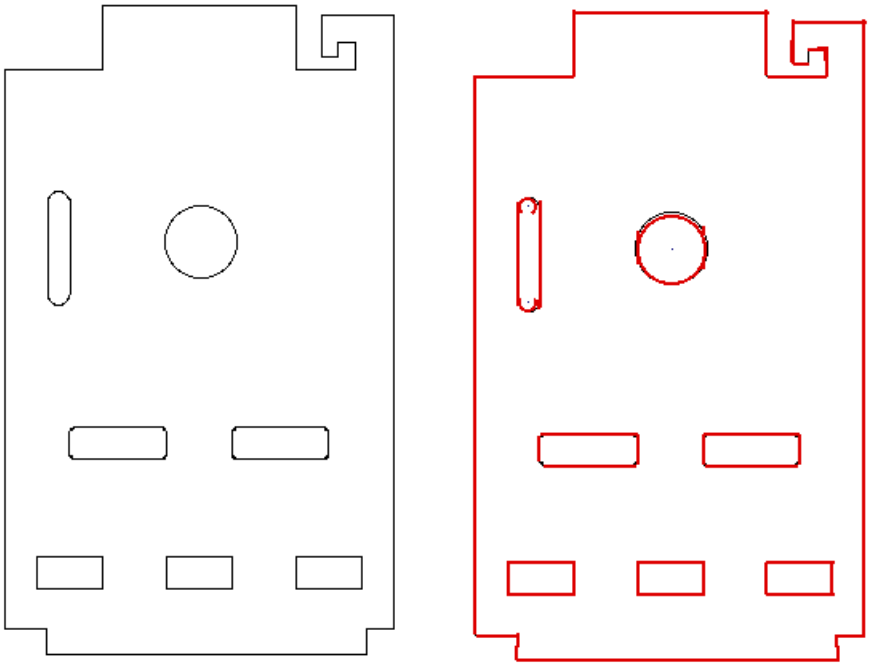

Figure 5. A fictional sheet metal pattern for a machine part

Appendix: Hierarchical Search in Parameter Space

Let us start by observing that the range of the parameter space is -1 to +1 in the k (line slope) dimension, and that the range in the c (intercept) dimension is a simple function of image size (it is determined by the values of the transforms of image corner-points, at $k = \pm 1$). The size of the parameter space is finite and manageable, and the entire space is the root quadrant of the search tree, as shown in Figure 3.

Every transform of an image point is a line of the form $c = Ak + B$, and the intersections of this line with the sides of the quadrant are fully determined by the values of the function $z = Ak - c + B$ at the corners. From the z-values at the corners of the parent quadrant we can determine which child quadrants the line intersects, but the calculation involves ratios and floating-point numbers.

We see from Figure 6 that, if z-values are known at the corners of child quadrants as well, we can determine whether the line intersects a side (sides are labeled a to l), merely by comparing the signs of z at the endpoints of that side. Since the z-values for at least three of the five corners of child quadrants would have to be calculated in the next iteration anyway, this is an acceptable overhead.

Furthermore, all dimensions can be scaled up by a power of two, the power being the maximum depth of the search quadtree. This converts all calculations to integer values, the sub-quadrant divisions by two amount to right shifts, and the ratios of z-values are

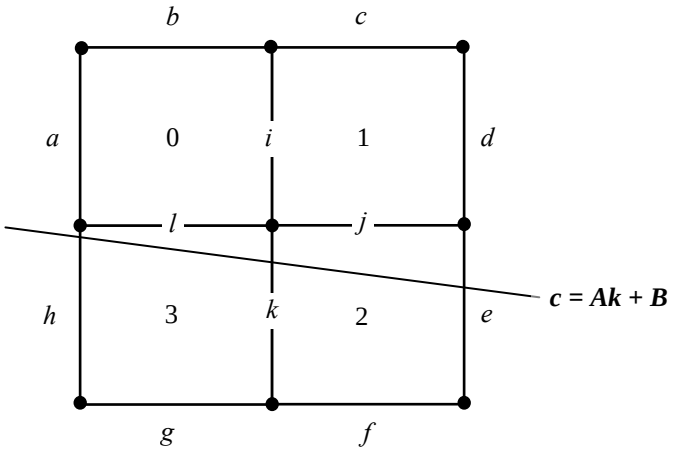

Figure 6. Point transform (a line) intersects the parent quadrant and a subset of child quadrants.

replaced with sign comparisons. This form of the search algorithm not only leads to fast code, but is also amenable to ASIC implementation.

In order to find which child quadrants the line intersects, we construct a decision tree, such as the one in Figure 7: inner nodes of this tree represent intersections with sides, and the leaf nodes yield all valid selections of child quadrants. Immediate question arises: what is the minimum depth of the decision tree, i.e. how many sides must be checked, in order to find the intersected child quadrants for an arbitrary line?

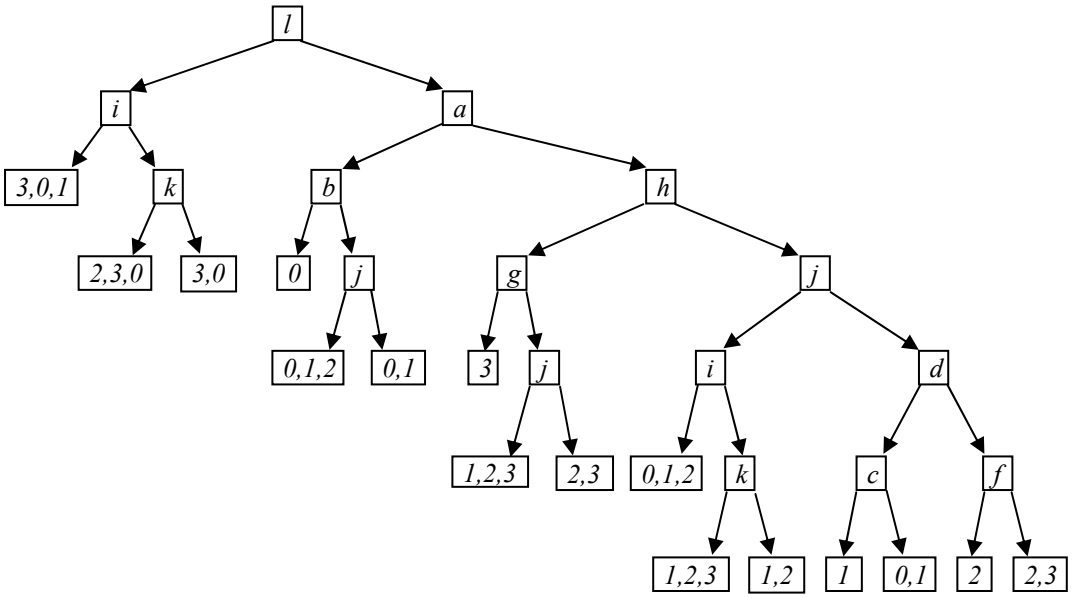

Figure 7. Decision tree: interior nodes represent quadrant sides (as in Figure 6); left arrows mean intersection of the line with a side, right arrows the lack of it. Leaf nodes are the selected sets of child quadrants. In a well-constructed tree, two intersections suffice to select a child set unambiguously.

Decision trees of depth six are easily constructed; in fact, there are great many of them. Are there trees of depth five? Following the rightmost path down any decision tree, we arrive at the quadrants whose intersecting lines do *not* intersect any of the sides chosen along that path (see example in Figure 8). It can be surmised by straightforward inspection that no choice of five non-intersectable sides is sufficient to select a unique set of child quadrants; an automated exhaustive search confirms this result. The rightmost path of any decision tree is always longer than five, therefore the least number of intersection decisions needed to uniquely select a set of child quadrants is six.

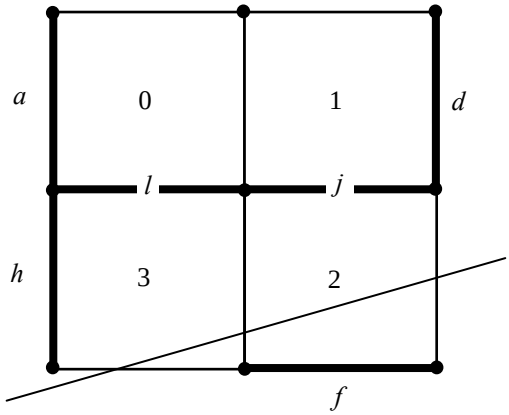

Figure 8. Quadrants 2 and 3 are unambiguously selected by lines which do not intersect any of the highlighted sides (rightmost path in the tree in Figure 7).

References

[1] Hough, P.V.C., "Method and Means for Recognizing Complex Patterns," U.S. Patent No. 3069654, 1962.

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

[3] Illingworth, J., Kittler, J., "The adaptive Hough transform," IEEE Trans. PAMI-9 (5), pp.690-698, 1987.

[4] Yuen, H.K., Princen, J., Illingworth, J., Kittler, J., "A comparative study of HT methods for circle finding," Proc. 5th Alvey Vision Conference, Manchester 1989.

[5] V.F. Leavers, "Shape Detection in Computer Vision Using the Hough Transform," Springer 1992.

[6] Leavers, V.F., "The dynamic generalized Hough transform …" CVGIP: Image Understanding, v.56(3), pp. 381-398, 1992.

[7] Ballard, D.H., "Generalizing the HT to detect arbitrary shapes," Pattern Recognition, v.13(2), pp.111-122, 1981.

[8] Stockman, G.C., Agrawala, A.K., "Equivalence of Hough curve detection to template matching," Comm. ACM v.20(11), pp. 820-822, 1977.

[9] Illingworth, J., Kittler, J., "A survey of the Hough transform," CVGIP 44, pp.87-116, 1988

[10] Davies, E.R., "Machine Vision: Theory, Algorithms, Practicalities," Academic Press 1990.

[11] Li, H., Lavin, M.A., LeMaster, R.J., "Fast HT: a hierarchical approach," CVGIP, v.36, pp.139-161, 1986.

[12] Lynx Auto-Digitizer, copyright of D. Antolovic, 1997. Indiana University has been given permanent right to use the source code of this application.

[13] T.Y. Zhang, C.Y. Suen, "A Fast Parallel Algorithm for Thinning Digital Patterns," Communications of the ACM, vol. 27, no. 3, pp. 236-239.

[14] R. Lumia, L. Shapiro and O. Zuniga, "A New Connected Components Algorithm for Virtual Memory Computers," Computer Vision, Graphics and Image Processing, v.22, pp.287-300, 1983.