

A New Approach for Circle Detection on Multiprocessors

A. Kavianpour , S. Shoari, and N. Bagherzadeh

University of California

Abstract

This paper considers the problem of detecting circles in images on multiprocessors. We have defined a new transformation that converts circles in an image to a families of straight lines allowing the problem to be converted to line detection which can be solved by any line detection algorithms. Also, we have developed two algorithms for circle detection based on this new transformation. A simulation program for these algorithms implemented on a pyramid architecture is presented.

Key Words: Circle detection, Hough transform, image processing, parallel processing, multiprocessors.

1 Introduction

The detection of lines and curves in an image is a fundamental problem in image processing. The problem often solved by Hough transform. Hough transform al-

gorithm was introduced by Paul Hough in a patent filed in 1962 [4]. Rosenfeld showed that it can be used to detect curves [9]. Later Duda and Hart [3] suggested that straight lines can be parameterized by the length ρ and orientation angle θ of the normal vector to the line from the image origin. This relationship is given by $\rho = x * \cos \theta + y * \sin \theta$. where ρ is the length of the line segment from the origin perpendicular to the line, and θ is the angle that the line makes with the positive x axis, measured clockwise.

For each point (x_i, y_i) the equation $\rho = x_i * \cos \theta + y_i * \sin \theta$ creates a set of sinusoidal curves on the (ρ, θ) plane as θ varies from 0 to π . Curves corresponding to the set of collinear figure points will have a common point of intersection represented as (ρ_i, θ_i) which defines the line connecting them in the Cartesian coordinates. A point in the Cartesian plane corresponds to a sinusoidal curve in the parameter plane, and a point in the parameter plane corresponds to a straight line in the Cartesian plane. When it is not necessary to determine the lines exactly, we can specify the acceptable error resolution ρ_{res} and θ_{res} . The parameter space ρ and θ can be quantized into $m * \rho$ values $\rho_1, \rho_2, \dots, \rho_m$ and $k * \theta$ angles, $\theta_1, \theta_2, \dots, \theta_k$ with an accumulator array of size $k * m$ each entry corresponds to a point defined by a pair of (ρ_i, θ_i) . A region is treated as a two-dimensional array of accumulator cells. For each point (x_i, y_i) in the Cartesian plane the (ρ, θ) of the cell selected by the curve with the relation $\rho = x_i * \cos \theta + y_i * \sin \theta$ is incremented. A given cell in the

two-dimensional accumulator array eventually records the total number of curves intersecting at the point represented by a given (ρ, θ) . After all the pixels have been treated, the parameter array is inspected to find those cells with the largest counts.

Recently several efforts have been made to speed up the computation of Hough transform by utilizing parallelism. Kavianpour and Bagherzadeh developed new and efficient algorithms on pyramid architectures for detecting curves using Hough transform [5, 6].

The application of Hough transform for circle detection in a digitized picture was considered by Duda and Hart [3]. In their algorithm a three dimensional array of accumulators was used. This array is indexed by three parameters specifying the location and size of a circle. This algorithm for real time application is prohibitively time consuming. Detection of approximate circles was considered by C. Kimme et al. [8]. The procedure is an extension and improvement of curve finding scheme described by Duda and Hart [3]. Casasent and Krishnapuram [2] have suggested a method which is based on the idea that a curve may be represented as a set of successive short line segments. For example, a circle can be represented by short straight line segments that are tangential to the circle at various points.

This paper presents our efforts in implementing an algorithm for detecting circles on multiprocessors. Section 2 describes a transformation which converts a circle to a line. Section 3 describes two algorithms for detecting circles, Section 4 explains

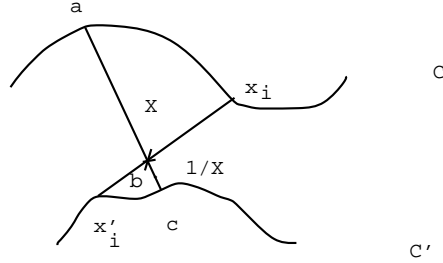


Figure 1: An inversive image of point a with respect to point b .

simulation program, and finally in Section 5 we present the conclusion.

2 Transformation from Circles to Lines

In this section, we will define a transformation that converts a circle to a line and hence facilitates circle detection algorithms. In the rest of this paper the notation $|ox|$ denotes the *absolute value* of a line ox i.e., the distance between points o and x .

Definition 1: Points a and c on the line abc are called *inversive image* of each other with respect to the *inversion point* b , if $|ab| * |bc| = 1$.

Figure 1 illustrates an inversive image property between points a and c with respect to point b . In this figure, points a and c are X and $1/X$ distance apart from the inversion point respectively. Based on Definition 1, the inversive property between two points which are inversive image of each other, is symmetric with respect to the inversion point.

Definition 2: An inversive image of a function C with respect to point b is another function C' such that for any point x_i on C there is an inversive image point x'_i on C' where relation $|bx_i| * |bx'_i| = 1$ is valid.

Figure 1 illustrates an inversive image of an arbitrary function C with respect to point b .

Definition 3: A pixel p_i is called a *test pixel* if it is selected as an inversion point of a binary image.

Theorem 1: An inversive image of a circle with respect to any point on the circle (i.e., the inversion point) is a straight line perpendicular to a line along the diameter passing through the inversion point.

Proof: Figure 2 illustrates a circle of diameter $|ab| = d$. Given point b represents the inversion point, let x_i be a point on the circle such that angle $\widehat{abx}_i = \theta$ where $-\pi/2 \leq \theta \leq \pi/2$. From the right-triangle $\triangle ax_ib$, we can conclude that $|x_ib| = d \cos \theta$ and inversive image of point x_i is a point (e.g., x'_i) such that $|bx'_i| = 1/|x_ib| = 1/(d \cos \theta)$. The projection of all points such as x'_i onto diameter ab constitute a line with distance $1/d$ from the inversion point b . This can be seen by the following relation in the $\triangle bcx'_i$ triangle.

$$(1/(d * \cos \theta)) * \cos \theta = 1/d.$$

Thus, inversive images of all points of a circle with respect to point b reside on a projection line $X = 1/d$ from point b .

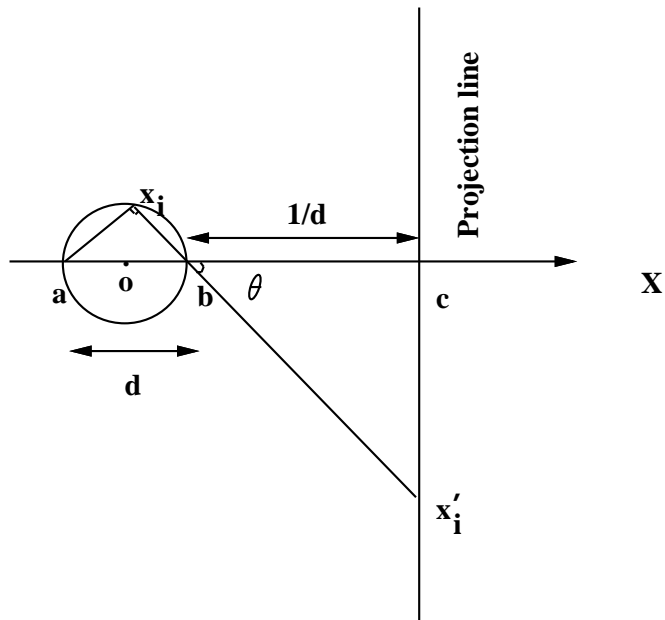


Figure 2: An inversive image of a circle with respect to point b .

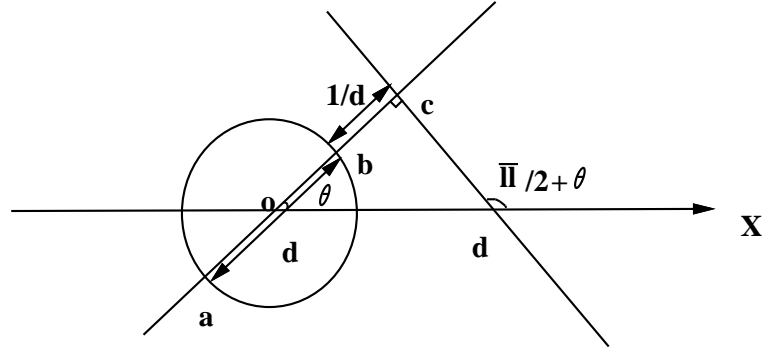


Figure 3: General illustration of the inversive image of a circle.

The following corollary follows from Theorem 1.

Corollary 1: An inversive image of a circle with respect to an inversion point that has an angle θ , $\theta \leq \pi/2$, with respect to the horizontal line passing through the center of circle is a straight line with an angle $\pi/2 + \theta$ (see Figure 3).

3 Circle Detection Algorithm

In this section circle detection algorithms on multiprocessors are considered. We assume that before the proposed circle detection algorithm is implemented the following low-level pixel processing operation have been applied to the binary image. (1)- Median filter - to remove unwanted black dots. (2)- Edge detection - to represent solid objects by their edges. (3)- Thinning - to reduce the edge width (from Step 2).

The procedure for detecting circles has two steps. In step one, coordinates of

inversive images will be calculated. In step two, Hough transform line detection algorithm will be applied.

3.1 General Circle Detection Algorithm(GCDA)

This algorithm describes a method for detecting any size circle in a binary image. We assume processor $p(i, j)$ stores $p_{x_{ij}}$ and $p_{y_{ij}}$, the x and y coordinates of the pixel at its local memory.

The four phases of GCDA algorithm are described below:

Phase one: Coordinates of the inversion point, test pixel p_{ij} , will be broadcast to all processors.

Phase two: In this phase each processor $p(k, l)$ computes the inversive image coordinates of its pixel p_{kl} with respect to inversion point, p_{ij} with $(i, j) \neq (k, l)$. $r_{x_{kl}}$ and $r_{y_{kl}}$ represent the coordinates of inversive image of the pixel p_{kl} and are calculated as follows:

From Figure 4 the following can be derived:

$$\sin \theta = (p_{y_{kl}} - p_{y_{ij}})/D_{kl} \text{ and } \cos \theta = (p_{x_{kl}} - p_{x_{ij}})/D_{kl}$$

$$r_{x_{kl}} = p_{x_{ij}} - (\cos \theta)/D_{kl} = p_{x_{ij}} - (p_{x_{kl}} - p_{x_{ij}})/D_{kl}^2$$

$$r_{y_{kl}} = p_{y_{ij}} - (\sin \theta)/D_{kl} = p_{y_{ij}} - (p_{y_{kl}} - p_{y_{ij}})/D_{kl}^2$$

$$\text{The value of } D_{kl} \text{ is equal to: } D_{kl} = \sqrt{(p_{x_{kl}} - p_{x_{ij}})^2 + (p_{y_{kl}} - p_{y_{ij}})^2}$$

After substitution we have the following:

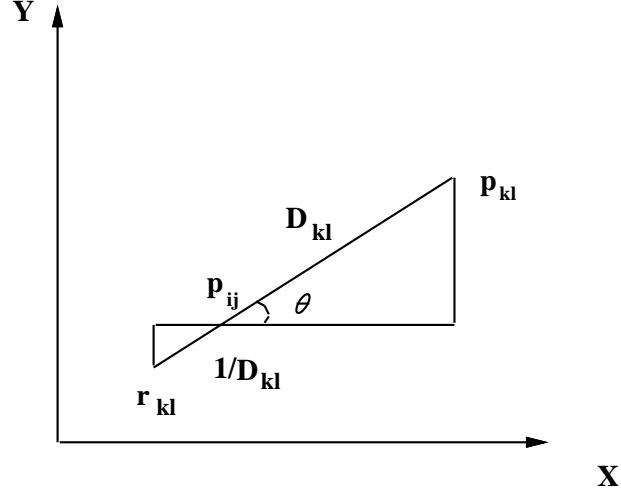


Figure 4: An inversive image coordinates of pixel p_{kl} with respect to test pixel p_{ij} .

$$r_{x_{kl}} = p_{x_{ij}} - (p_{x_{kl}} - p_{x_{ij}}) / ((p_{x_{kl}} - p_{x_{ij}})^2 + (p_{y_{kl}} - p_{y_{ij}})^2), \text{ and}$$

$$r_{y_{kl}} = p_{y_{ij}} - (p_{y_{kl}} - p_{y_{ij}}) / ((p_{x_{kl}} - p_{x_{ij}})^2 + (p_{y_{kl}} - p_{y_{ij}})^2).$$

These coordinates constitute all the points that have to be checked for line detection.

Phase three: Inversive image coordinates with respect to pixel p_{ij} are saved and a new test pixel p'_{ij} is selected for repeating the algorithm.

Phase four: In this phase Hough transform line detection algorithm i.e., the one described in our previous work [5] will be applied. At the end of this phase the coordinates of the circle for test pixel p_{ij} is calculated.

Phases one through four are repeated for all test pixels p_{ij} .

3.2 Specific Circles Detection Algorithm(SCDA)

This algorithm describes a method for detecting circles of diameter d_i for $i = 1, 2, \dots, m$ in a binary image. We assume processor $p(i, j)$ stores $p_{x_{ij}}$ and $p_{y_{ij}}$, the x and y coordinates of the pixel, and also it stores d_i , diameter of the circle at its local memory.

The SCDA algorithm has five phases as described below:

Phase one: Coordinates of the inversion point, test pixel p_{ij} , will be broadcast to all processors.

Phase two: In this phase each processor $p(i, j)$ computes the distance of its pixel p_{kl} from test pixel p_{ij} , by calculating D_{kl} .

Phase three: Each processor compares the computed distance D_{kl} with diameter d_i . Those processors that satisfy $D_{kl} \leq d_i$ will calculate the inversive image coordinates of their pixel p_{kl} with respect to the test pixel, p_{ij} . Those processors that do not satisfy $D_{kl} \leq d_i$, will receive new test pixel. $r_{x_{kl}}$ and $r_{y_{kl}}$ represent the coordinates of inversive image of the pixel p_{kl} and are calculated using the same equations as in the Phase two of GCDA algorithm (see Figure 4). These coordinates constitute all the points that have to be checked for line detection.

Phase four: The coordinates of inversive image are saved and a new test pixel p'_{ij} is selected for repeating the algorithm.

Phase five: In this phase Hough line detection algorithm i.e., the one described in our previous work [5] will be applied to detect lines that are at distance $1/d_i$ from test pixel. At the end of this phase the coordinates of the circle for the test pixel p_{ij} is calculated.

Phases one through five are repeated for all test pixels p_{ij} . Similar to GCDA algorithm, SCDA allows for pipeline operations at each phase.

4 Simulation Program

In order to test the proposed algorithms on an architecture, we selected a simulation environment that tests the performance of the algorithms on a pyramid architecture, however, this method could be applied to the other pixel-based architectures such as 2D mesh and hypercube. The simulation program for detecting circles was written in C language [7, 1]. Figure 5 illustrates a 16 by 16 image created by *bitmap editor* under X windows [10].

The bitmap file of the image in Figure 5 is illustrated in the Figure 6 and is called portable bitmap(pbm). In this pbm file 1 means *black* and 0 means *white*. The simulation program consists of two major parts: *Simulation Controller* and *Algorithm Controller*. The Simulation Controller is used for scheduling simulation tasks. It offers a menu to the user to select the size of a pyramid. The task of Algorithm Controller is to select one of the image processing algorithms, i.e., GCDA or SCDA.

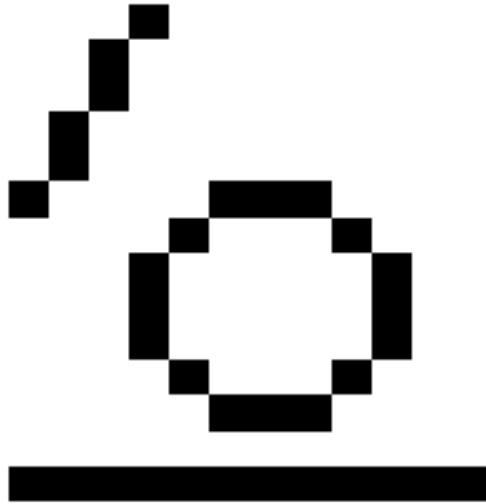


Figure 5: The 16x16 image created by bitmap editor.

The program has a global clock called *Simulation Clock* and incremented whenever the processors at the same level terminate execution. Therefore, at the end of the program, Simulation Clock denotes the number of times that processors activated during execution of the program. In order to make the circle detection algorithms more efficient, only for *black* pixels inversive points are calculated. Therefore, the simulation clock of the program depends on the number of *black* pixels in the image.

Table 1 illustrates the computer simulation results for different size binary images using GCDA and SCDA algorithms.

Figure 7 represents the relationship between image size and simulation clock for GCDA and SCDA algorithms.

```

0000000000000000
0000100000000000
0000100000000000
0000100000000000
0001000000000000
0001000000000000
0010000111000000
0000001000100000
0000010000010000
0000010000010000
0000010000010000
0000001000100000
0000000111000000
0000000000000000
0011111111111100
0000000000000000

```

Figure 6: The pbm file of a 16x16 image.

<i>Image size</i>	<i>GCDA</i>	<i>SCDA</i>
8 <i>by</i> 8	568	376
16 <i>by</i> 16	5,808	4,112
32 <i>by</i> 32	54,432	38,992
64 <i>by</i> 64	322,176	233,280
128 <i>by</i> 128	2,428,800	1,670,144
256 <i>by</i> 256	13,759,488	9,554,688
512 <i>by</i> 512	84,300,288	60,990,400

Table 1: Simulation clock for circle detection algorithms

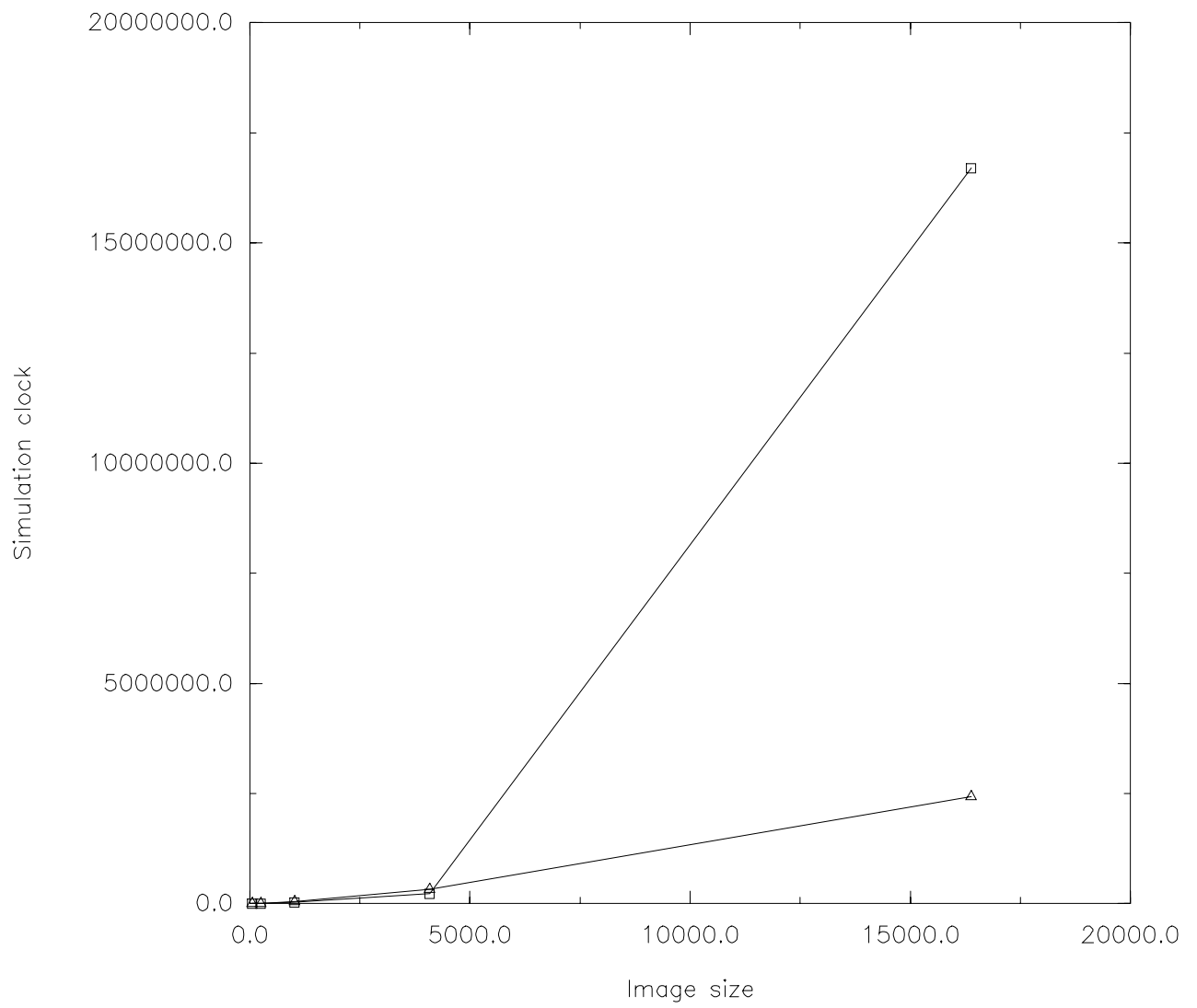


Figure 7: Simulation clock for circle detection algorithms

5 Conclusion

The problem of detecting circles in images on multiprocessors is considered. We have defined a new transformation that converts circles in an image to a families of straight lines allowing the problem to be converted to line detection which can be solved by any line detection algorithms, i.e., Hough transform. Also, we have developed two algorithms for circle detection. A simulation program for evaluating these algorithms is presented.

Acknowledgement: The work presented was supported in part by the UC MICRO program under Grant No. 89-108 in cooperation with Rockwell International. Additional funds were supplied by Irvine Faculty Research Fellowship.

References

- [1] Go Beale, "Real-time simulation of dynamic systems on a pyramid architecture," *IEEE Trans. on Industrial Electronics*, vol. 37, no. 3, pp.212-220, June 1990.
- [2] D. Casasent and R. Krishnapuram, "Curve Object Location by Hough Transformations and Inversions," *Pattern Recognition*, vol. 20, No. 2, pp. 181-188, 1987.

- [3] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *CACM*, 15, (1), pp. 11-15, 1972.
- [4] P. V. C. Hough, "A method and means for recognizing complex patterns," U.S. Patent 3,069,654, 1962.
- [5] A. Kavianpour and N. Bagherzadeh, "Parallel Hough Transform for Image Processing on a Pyramid Architecture," *International Conference on Parallel Processing*, pp. 395-398, August 1991.
- [6] A. Kavianpour, and N. Bagherzadeh, "Circle Detection in Black and White Images," *Patent pending* UC Case No. 91-195-1, 1991.
- [7] Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, Prentice Hall, Englewood Cliffs, New Jersey, 1978.
- [8] C. Kimme, D. Ballard, and J. Skalansky, "Finding Circles by an array of Accumulators," *Communication of ACM*, vol. 18, pp. 120-122, 1975.
- [9] A. Rosenfeld, *Picture processing by computer*, Academic Press, 1969.
- [10] Unix on-line computer manual.