TR-88-21

BRESENHAM'S ALGORITHM IN QUADCODES

Bresenham's Algorithm in Quadcodes

J. P. Hwang and K. Y. Cheng

Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan, R.O.C.
and
Institute of Information Science
Academia Sinica

ABSTRACT

An algorithm for drawing line-segment between two pixels in a binary raster image coded in linear quadtree representation is presented. It is a Bresenham's algorithm in quadcodes and is as efficient as the original Bresenham's algorithm when implemented in hardware.

1. Introduction

Using quadtree to represent image data structures has gained its popularity in dealing with raster scan graphics recently [1,2,3,4,5,6,7]. The major advantage of using quadtree representation lies in the fact that the image pixel data structured in a recursive quadrant-division-tree can provide parallel computation on extracting local image information. However, image data coordinates are lost in the quadtree representation, because now the pixel data are in terms of strings of codes. Therefore, a conversion process to transform codes into their corresponding image coordinates must be provoded when using quadcodes in graphics.

In this paper, we present a method that uses quadcodes in Bresenham's algorithm. Bresenham's algorithm [8] has been accepted as a standard approach to draw any line segment between two points in raster scan graphics. As stated previously, in order to extend the application of Bresenham's algorithm to the image structured in quadcodes, we have to find the adjacent relation of quadcodes as well as the distance between quadcodes in the generation of the line segment. In the following, we shall divide our discussion into sections. Section 2 briefly introduces the definition of quadcode. Section 3 shows the formula for neighbors finding and distance measure between quadcodes. Section 4 shows the original Bresenham's algorithm. Section 5 demonstrates portions of the original algorithm which need be converted into quadcode expression and then shows the complete Bresenham's algorithm in quadcodes.

2. Quadcode Definition

The quadcode is a quaternary-based string code. Let S = { 0, 1, 2, 3 } be the code character set. Then a quadcode of length n is of the form

$$Q = q_1 q_2 \cdots q_n$$

where $q_i \varepsilon S$ for $i = 1, 2, ..., n$.

The code characters are arranged into quadrants as shown in Figure 2.1. Each quadcode character represents an operation of subdividing the image or its subimage into quadrants. The continuation of the subdivision is represented in a quadtree structure [9,10]. And, coded similar to the linear quadtree approach [11], a quadcode represents a region of square quadrant with black pixels [12]. Hence a quadcode represents a black node of a linear quadtree, i.e., a region which needs no further subdivision in that particular quadrant. Figure 2.2 shows an example of a $2^3 \times 2^3$ image represented in quadtree, the quadcodes are 030, 031, 033, 12, 132, 2, 30, and 32, they represent the regions : R, S, U, L, X, D, N, and P, repectively.

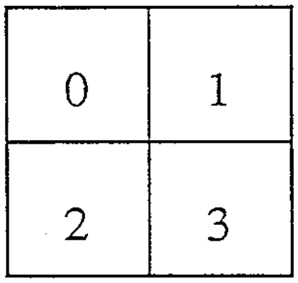
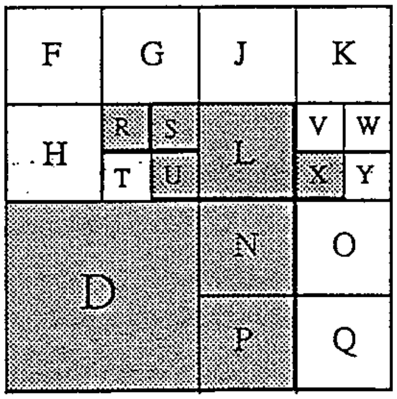


Figure 2.1. Quadcode characters in quadrants.



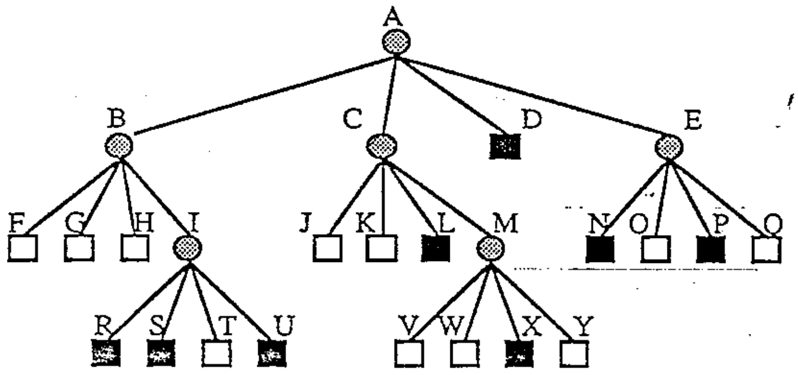


Figure 2.2. Quadtree representation of a $2^3 x 2^3$ image.

3. Quadcode Relations

3.1 Neighbors Finding

To each quadcode, its eight adjacent neighbors (of same size) can be obtained from the detection of adjacent relations. For example, consider the node N_a with quadcode $a_1a_2...a_{n-1}0$ and two of its neighbors N_r and N_u shown in Figure 3.1. Suppose their quadcodes are each of length n. As can be seen, the father node of both N_r and N_a is $N_{fa} = (a_1a_2...a_{n-1})$, thus $N_r = a_1a_2...a_{n-1}1$. However, the father node of N_u is N_{fu} which is different from that of N_a. But we see that the last code character of N_u must be 2, so $N_u = u_1u_2...u_{n-1}2$. Also, the direction of N_{fu} relative to N_{fa} is an above relation. This observation suggests us to use a row function $t_a(0,d_j) = (a_i',d_j')$ where $d_j = \uparrow, \downarrow, \leftarrow, \rightarrow, \nwarrow, \nearrow, \swarrow, \searrow$, $a_i' = 1, 2, 3$, and $d_j' = \uparrow, \leftarrow, \nwarrow, \phi$ repectively, to denote the observed adjacency relations. With the same reasoning, an adjacent relation table $t_a(a_i,d_j) = (a_i',d_j')$, $i = 1,...,4$ and $j = 1,...,8$, can be constructed also. The adjacent relation table which indicates both the location of neighbors' ancestor (or descendant) nodes as well as the direction to search them is shown in Figure 3.2.

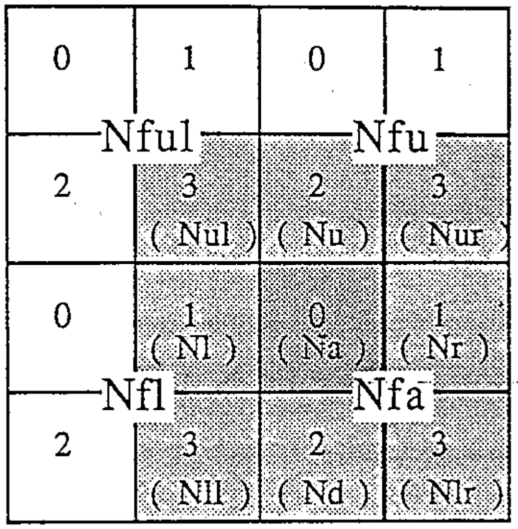


Figure 3.1. Adjacency relations between N_a and its eight neighbors.

	↑	↓	←	→	↖	↗	↙	↘
0	(2,↑)	(2,ϕ)	(1,←)	(1,ϕ)	(3,↖)	(3,↑)	(3,←)	(3,ϕ)
1	(3,↑)	(3,ϕ)	(0,ϕ)	(0,→)	(2,↑)	(2,↗)	(2,ϕ)	(2,→)
2	(0,ϕ)	(0,↓)	(3,←)	(3,ϕ)	(1,←)	(1,ϕ)	(1,↙)	(1,↓)
3	(1,ϕ)	(1,↓)	(2,ϕ)	(2,→)	(0,ϕ)	(0,→)	(0,↓)	(0,↘)

Figure 3.2. The adjacent relation table.

Using this adjacent relation table, we can start from a node N_a with quadcode $a_1a_2...a_n$ to find its eight adjacent neighbors by the following (recursive) formula (3.1):

(3.1) $AN(a_1a_2...a_n, d_j) =$

$(a_1a_2...a_{n-1})(A')$, if $n > 0$ and $D' = \phi$,

$$(AN(a_1a_2...a_{n-1},D'))(A') \quad , \text{if } n > 0 \text{ and } D' \neq \phi,$$

$$Nil \quad , \text{if } (AN(\varepsilon, d_j))(b_1b_2...b_n) \text{ is reached.}$$

where $A' = t_a(a_n,d_j).a_n'$ and $D' = t_a(a_n,d_j).d_j'$.

The following procedure uses formula (3.1) to find an appropriate neighbor. We propose two versions, recursive and nonrecursive, in this procedure.

```
Procedure 3.1: Neighbor
    Input:  $N = a_1a_2...a_n$ and $Dir = \uparrow, \downarrow, \rightarrow, \leftarrow, \nwarrow, \nearrow, \swarrow$, or $\searrow$
    Output: The neighbor of N in direction Dir

    /* Recursive version */
    if $t_a(a_n,Dir).d_j' = \phi$ then
        return($(a_1a_2...a_{n-1})(t_a(a_n,Dir).a_i')$)
    else
        if $n = 1$ then
            return(Nil)
        else
            return($(Neighbor(a_1a_2...a_{n-1},t_a(a_n,Dir).d_j'))(t_a(a_n,Dir).a_i')$)
        end if
    end if

    /* Nonrecursive version */
    $D = Dir$
    for $p = n$ down to 1
        if $t_a(a_p,D).d_j' = \phi$ then
            return($(a_1a_2...a_{p-1})(t_a(a_p,D).a_i')(b_{p+1}b_{p+2}...b_n)$)
        else
            $b_p = t_a(a_p,D).a_i'$
            $D = t_a(a_p,D).d_j'$
        end if
    next p
    return(Nil)
```

3.2 Distance Measure

In order to measure the distance between two quadcodes, we need to perform both the horizontal and vertical projections to calculate the X- and Y-components of the distance. The horizontal projection horizontally moves the quadcode node to the leftest position, while the vertical projection vertically moves the quadcode node to the uppermost position. For example, the projections of two nodes N_a (123) and N_b (213) of a $2^3 x 2^3$ image in Figure 3.3 obtain N_{ha} (022), N_{hb} (202), N_{va} (101), and N_{vb} (011) repectively.

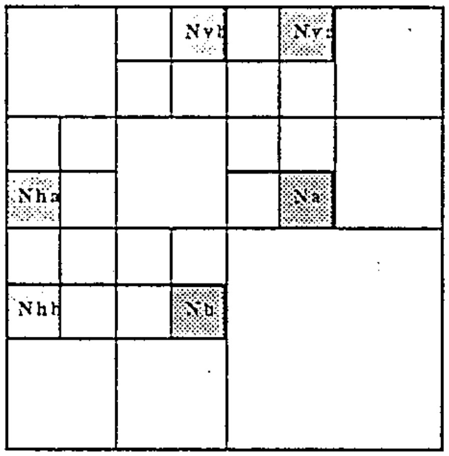


Figure 3.3 Horizontal and vertical projections.

Notice that each code character in the horizontal projection is either 0 or 2, we have to change them to 0 or 1 respectively. This can be obtained by dividing each original code character by 2. While each code character in the vertical projection is either 0 or 1, we need not change them, and this can be obtained by operating to each original code character with mod. 2. Based on the above observation, we have the following two procedures for horizontal and vertical projections :

Procedure 3.2: HP (Horizontal Projection)
Input: $N = a_1 a_2 ... a_n$
Output: $N_h = h_1 h_2 ... h_n$ (the quadcode after horizontal projection)

```
for i = 1 to n
    $h_i = a_i / 2$
next i
return($h_1 h_2 ... h_n$)
```

Procedure 3.3: VP (Vertical Projection)
Input: $N = a_1 a_2 ... a_n$
Output: $N_v = v_1 v_2 ... v_n$ (the quadcode after vertical projection)

```
for i = 1 to n
    $v_i = a_i \bmod 2$
next i
return($v_1 v_2 ... v_n$)
```

To measure the distance between two quadcodes, we perform the horizontal projection to find its vertical component and the vertical projection to find its horizontal component. Hence the distance can be calculated as follows: $\text{Distance}(N_b,N_a) = \text{Sqrt}\{[\text{Diff}(\text{VP}(N_b),\text{VP}(N_a))]^2 + [\text{Diff}(\text{HP}(N_b),\text{HP}(N_a))]^2\}$, where Diff is obtained from the following procedure:

Procedure 3.4: Diff (horizontal or vertical distance component)
Input: $N_a = a_1 a_2 ... a_n$ and $N_b = b_1 b_2 ... b_n$ (projected nodes)
Output: The distance between N_a and N_b, or $N_a - N_b$.

```
d = 0
for i = 1 to n
    $d = d * 2 + (a_i - b_i)$
next i
return(d)
```

4. Original Bresenham's Algorithm

Bresenham's algorithm seeks to find the raster locations of a line segment between two points (x_1,y_1) and (x_2,y_2). The basic idea of this algorithm is to incrementally seek a neighbor (one unit a time) in either X or Y depending on the slope of the line. Suppose one unit of a line segment is in the first octant as shown in Fig. 4.1. The increment is either 0 or 1 is determined by examining the distance between the actual line location and the nearest grid location (1,0) and (1,1). The distance between the actual line location and (1, 0) is called the error. If error $\geqq 0$ then plot (1,1) and update the error value to the distance between the actual line location and (1,1), else if error < 0 then polt (1,0) and propagate the error value to the next unit. This process is repeated until (x_2,y_2) is reached. For later discussion, the original Bresenham's algorithm is shown as follows:

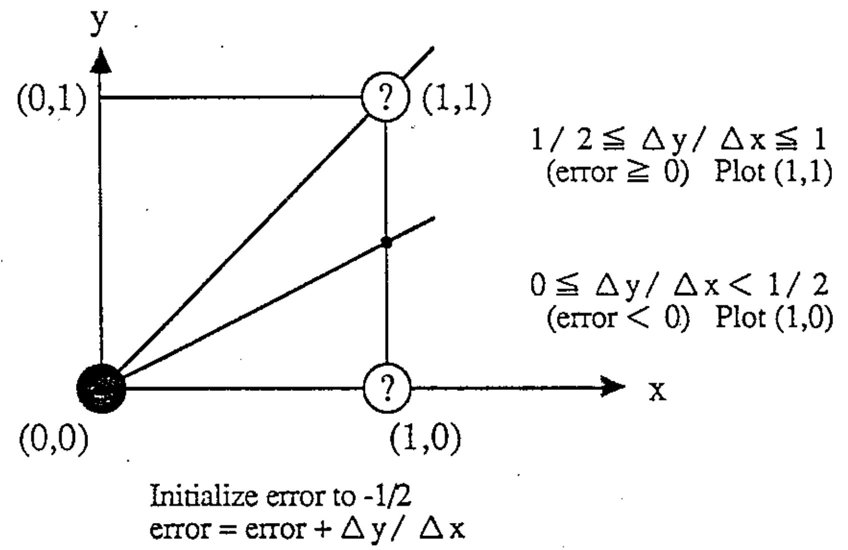

y
(0,1)
?
(1,1)
$1/2 \leqq \Delta y/\Delta x \leqq 1$
(error $\geqq$ 0) Plot (1,1)
$0 \leqq \Delta y/\Delta x < 1/2$
(error < 0) Plot (1,0)
?
x
(0,0)
(1,0)
Initialize error to -1/2
error = error + $\Delta y/\Delta x$

Figure 4.1. Basis of Bresenham's algorithm.

```
Original Bresenham's algorithm:
/*  the line end points are $(x_1,y_1)$ and $(x_2,y_2)$ assumed not equal
    all variables are assumed integer
    the Sign function returns -1, 0, 1 as its argument is < 0, = 0, or > 0 */

/*  initialize variables */
    $x = x_1$
    $y = y_1$
    $\Delta x = abs(x_2 - x_1)$
    $\Delta y = abs(y_2 - y_1)$
    $s_1 = \text{Sign}(x_2 - x_1)$
    $s_2 = \text{Sign}(y_2 - y_1)$

/*  Interchange $\Delta x$ and $\Delta y$ depending on the slope of the line */
    if $\Delta y > \Delta x$ then
        Temp = $\Delta x$
        $\Delta x = \Delta y$
        $\Delta y$ = Temp
        Interchange = 1
    else
        Interchange = 0
    end if

/*  Initialize the error term to compensate for a nonzero intercept */
    $e = 2 * \Delta y - \Delta x$

/*  main loop */
    for i = 1 to $\Delta x$
        Plot(x,y)
        if $e \geqq 0$ then
            $x = x + s_1$
            $y = y + s_2$
            $e = e - 2 * (\Delta x - \Delta y)$
        else
            if Interchange = 1 then
                $y = y + s_2$
            else
                $x = x + s_1$
            end if
            $e = e + 2 * \Delta y$
        end if
    next i
    finish
```

Notice that the sign s_i, $i = 1, 2$, are determined by what quadrant the line segment lies, as shown in Figure 4.2. For example, if the line is in the first quadrant then $s_1 = x_2 - x_1 > 0$ and $s_2 = y_2 - y_1 > 0$. Also notice that in each quadrant, there are two regions seprated by a slope line. These slope lines are used to indicate conditions for the interchange of Δx and Δy. For instance, note that in the first quadrant, if the slope angle is less than 45° then x is incremented by 1 else y is incremented by 1.

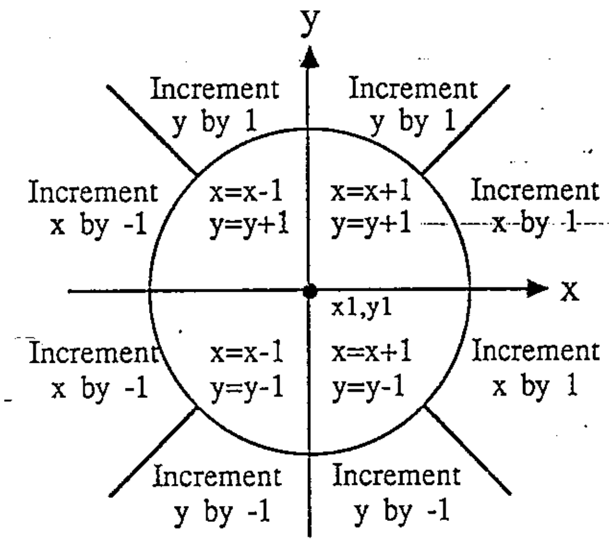


Figure 4.2. Conditions for original Bresenham's algorithm.

5. Bresenham's algorithm in Quadcodes

In order for using quadcodes in the implementation of Bresenham's algorithm, few things need be considered. First, the increment in x or y must be changed to a searching direction for finding a neighbor. Fig. 5.1 shows these searching directions in four quadrants. For example, suppose the line segment is in the lower region of the first quadrant (or the first octant) and if the plot is (1,1) then the searching direction is NE (North-East) or if (1,0) then E (East). Observe that there are only two searching directions in each octant, the diagonal and coordinate directions. The procedures to find these directions can be stated as follows:

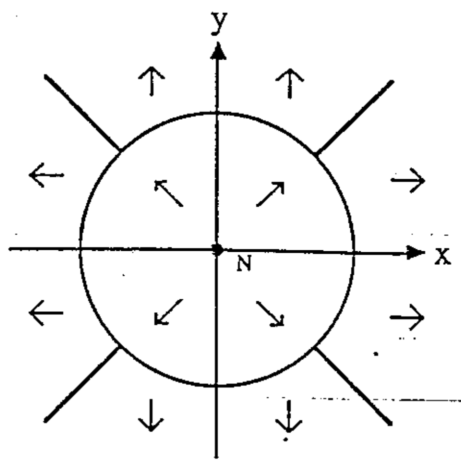


Figure 5.1. Conditions for Bresenham's algorithm in quadcodes.

Procedure 5.1: Diagonal_Direction
Input: Δx and Δy.
Output: The diagonal searching direction ↖ or ↗ or ↙ or ↘ or don't-care, depending on the signs of Δx and Δy.

```
if ( $\Delta x > 0$ and $\Delta y > 0$) then return(↗) end if
if ( $\Delta x < 0$ and $\Delta y > 0$) then return(↖) end if
if ( $\Delta x < 0$ and $\Delta y < 0$) then return(↙) end if
if ( $\Delta x > 0$ and $\Delta y < 0$) then return(↘) end if
return(don't-care)  /* $\Delta x = 0$ or $\Delta y = 0$ */
```

Procedure 5.2: Coordinate_Direction
Input: Δx and Δy.
Output: The coordinate searching direction ↑ or ↓ or →, or ← or don't-care, dependng on the signs of Δx and Δy and the slope of the line.

```
if ( $\Delta y > 0$ and abs($\Delta y$) > abs($\Delta x$)) then return(↑) end if
if ( $\Delta y < 0$ and abs($\Delta y$) > abs($\Delta x$)) then return(↓) end if
if ( $\Delta x > 0$ and abs($\Delta x$) > abs($\Delta y$)) then return(→) end if
if ( $\Delta x < 0$ and abs($\Delta x$) > abs($\Delta y$)) then return(←) end if
return(don't-care)      /* abs($\Delta x$) = abs($\Delta y$) */
```

Let the initial variables $x = x_1$ and $y = y_1$ be $N = N_a$ and (x_2,y_2) be N_b. Since in the original algorithm, Δx and Δy are vertical and horizontal distance components repectively and we use $\text{Diff}(N_1,N_2)$ to denote a projection distance, then $\Delta x = \mathrm{Diff}(VP(N_b), VP(N_a))$ and $\Delta y = \mathrm{Diff}(HP(N_b), HP(N_a))$.

As stated previously, sign s_i, $i = 1, 2$, must be changed to the searching direction when using quadcodes. Since at each octant, there are diagonal and coordinate directions only, we use Dird = Diagonal_Direction(Δx,Δy) and Dirc = Coordinate_Direction(Δx,Δy) to denote the diagonal and coordinate searching directions respectively. Based on these modification, the complete Bresenham's algorithm in quadcodes can be shown below:

```
Bresenham's algorithm in quadcodes:
/*  the line end points are $N_a$ and $N_b$ assumed not equal */

/*  initialize variables */
    N = Na
    $\Delta x = \mathrm{Diff}(VP(N_b),VP(N_a))$
    $\Delta y = \mathrm{Diff}(HP(N_b),HP(N_a))$
    Dird = Diagonal_Direction($\Delta x$,$\Delta y$)
    Dirc = Coordinate_Direction($\Delta x$,$\Delta y$).
    $\Delta x = \mathrm{abs}(\Delta x)$
    $\Delta y = \mathrm{abs}(\Delta y)$

/*  Interchange $\Delta x$ and $\Delta y$ depending on the slope of the line */
    if $\Delta y > \Delta x$ then
        Temp = $\Delta x$
        $\Delta x = \Delta y$
        $\Delta y$ = Temp
    end if

/*  Initialize the error term to compensate for a nonzero intercept */
    $e = 2 * \Delta y - \Delta x$

/*  main loop */
    for i = 1 to $\Delta x$
        Plot(N)
        if $e \geqq 0$ then
            N = Neighbor(N,Dird)
            $e = e - 2 * (\Delta x - \Delta y)$
        else
            N = Neighbor(N,Dirc)
            $e = e + 2 * \Delta y$
    end if
next i
finish
```

6. Conclusion

In this paper, we present a quadcode version of Bresenham's algorithm. Although from what we presented, this version seems not as efficient as the original one in computational complexity. However, when the quadcode algorithm is implemented in hardware then they have the same computational complexity. The reason is as follows. Suppose the image pixel location x and y are coded as $x_{n-1}x_{n-2}\cdots x_0$ and $y_{n-1}y_{n-2}\cdots y_0$ binary bit strings repectively. Then the corresponding quadcode string will be $\ulcorner y_{n-1}x_{n-1}\lrcorner\ \ulcorner y_{n-2}x_{n-2}\lrcorner \cdots \ulcorner y_0x_0\lrcorner$, where $\ulcorner\ \lrcorner$ denotes a quadcode character. Hence the neighbor finding can be arranged into a unit increment circuit such that a neighbor finding in a coordinate searching direction is an incremental $x + 1$ or $y + 1$ in the original algorithm. Thus the neighbor finding implemented in this way requires one operation only. Also, to the extend of authors' knowledge, there seems no quadcode version of Bresenham's algorithm has ever been published, we believe this paper can fill up this vacancy.

REFERENCES

[1] C. H. Chien and J. K. Aggarwal, "A Normalized Quadtree Representation," *Comput. Vision, Graphics, Image Process.* 26, 1984, pp. 331-346.

[2] I. Gargantini, "Translation, Rotation, and Superposition of Linear Quadtrees," *Int. J. Man-Machine Studies*, Vol. 18, 1983, pp. 253-263.

[3] W. I. Grosky and R. Jain, "Optimal Quadtrees for Image Segments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-5, No. 1, 1983, pp. 77-83.

[4] E. Kawaguchi and T. Endo, "On a Method of Binary Picture Representation and its Application to Data Compression," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-2, 1980, pp. 27-35.

[5] D. M. Mark and D. J. Abel, "Linear Quadtrees from Vector representations of Polygons", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-7, No. 3, 1985, pp. 344-349.

[6] M. Shneier, "Calculations of Geometric Properties Using Quadtrees," *Computer Graphics and Image Processing*, Vol. 16, 1981, pp. 296-302.

[7] J. R. Woodmark, "The Explicit Quad Tree as a Structure for Computer Graphics," *The Computer Journal*, Vol. 25, No. 2, 1982, pp. 235-238.

[8] Bresenham, J. E., "Algorithm for Computer Control of a Digital Plotter," *IBM System Journal*, Vol. 4, 1965, pp. 25-30.

[9] G. M. Hunter and K. Steiglitz, "Operations on Image Using Quadtrees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-1, No. 1, 1979, pp. 50-60.

[10] H. Samet, "An Algorithm for Converting Rasters to Quadtrees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-3, No. 1, 1981, pp. 93-95.

[11] I. Gargantini, "An Effective Way to Represent Quadtrees," *Communications of the ACM*, Vol. 25, No. 12, 1982, pp. 905-910.

[12] Shu-Xiand Li and M. H. Leow, "The Quadcode and its Arithmetic," *Communications of the ACM*, Vol. 30, No. 7, 1987, pp. 621-626.