M.Tech. (Computer Science) Dissertation Series

# Parallel Architecture and Algorithm for Fast Detection of Circles in 2D Images

A dissertation submitted in partial fulfilment of the requirements for the M.Tech (Computer Science) degree of Indian Statistical Institute, Calcutta.

*By*

**T. Sudhakar Rao**

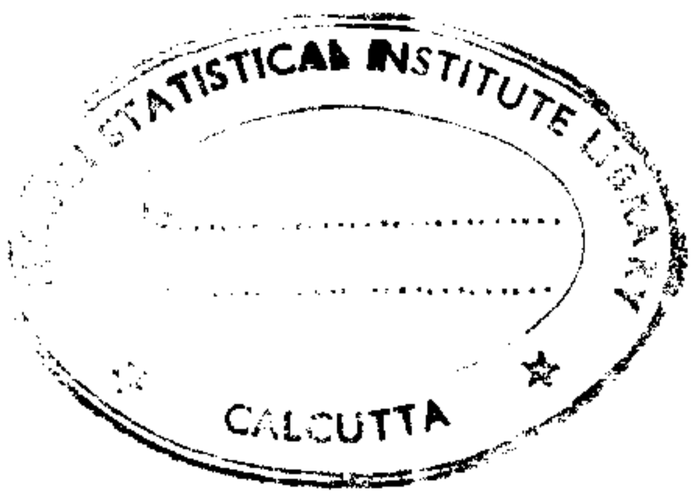
Under the supervision of

**Prof. Bhabani P. Sinha**

**INDIAN STATISTICAL INSTITUTE**

203, Barrackpore Trunk Road

Calcutta – 700035




July 1998

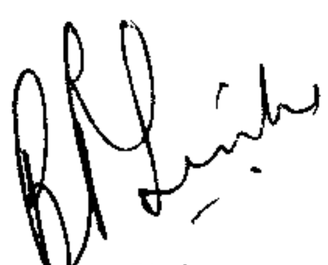
**Indian Statistical Institute**

**203, Barrackpore Trunk Road**

**Calcutta – 700035**

## *Certificate of approval*

This is to certify that the thesis titled, **Parallel architecture and algorithm for fast detection of circles in 2D images** submitted by **T.Sudhakar Rao** towards the partial fulfillment of requirements for the M.Tech (Computer Science) degree of Indian Statistical Institute, Calcutta, embodies the work done under my supervision.



Bhabani P. Sinha

Professor and Head

Advanced Computing and Microelectronics Unit

Indain Statistical Institute

Calcutta- 700035

# Abstract

Machine perception of scene data is an important application of Computer Vision. Parallel Processing may reduce it's time complexity considerably. In this dissertation we try to simulate a part of the fast, but approximate and adaptive pattern matching characteristics of human brain. A novel parallel architecture using content addressable memory for fast identification of continuous circles in an image was devised. If the image consists of only circles, the algorithm runs in $O(N)$ time to identify all circles in a 2D image, where $N$ is the number of circles in the image.

# Acknowledgments

I take pleasure in expressing gratitude to my guide Prof. B.P.Sinha for his guidance, advice, encouragement. I am thankful to Dr. Mallika De for her constructive criticism. I offer thanks to all the staff of Advanced Computing and Microelectronics Department.

I thank all my classmates especially P.B.K.Reddy and G.Surekha for helping me during my stay of two years in ISI. I thank Parveen Gupta M.Tech,C.S.-I Year for allowing me to use his computer.

Calcutta,
July 1998.

(T.Sudhakar Rao)

# Contents


1. Introduction *1*

2. Problem Description & Assumptions *3*

3. Categories of Illuminated Pixels *6*

4. Curvature *11*

5. Architecture *12*

6. Data Structures *14*

7. Algorithm 16

   **7.1** Algorithm *16*

   **7.2** Complexity 20

8. Conclusions *21*

References *22*

Appendix *23*

Chapter 1

# Introduction

Identification of features of objects from given images is an important problem in the field of Computer Vision. Still the time taken in this process by the machine is large compared to that of human brain. The human brain takes a lot of time for computations involving number crunching but recognizes any object or pattern very quickly in spite of its slow response to the stimuli; A computer on the contrary is very fast in numerical calculations but slow in recognizing any pattern or object.

The human brain is basically an adaptive pattern recognizing system. Matching of patterns by human brain is performed in an hierarchical structure. First the highest level details, later the lower level details are considered for matching. This process is continued until the level of matching attains some degree of confidence or a number of failures (mismatch) are encountered. The time and number of levels involved in the matching process is dependent on the maturity level of brain.

The above behavior of human brain can be simulated by a model based on a pattern recognizing architecture. In this model any operation is performed by comparing it with a fixed set of previously known objects or patterns. However this domain of fixed stored patterns will increase in gradual learning process. Based on this model, fast parallel algorithms for detection of straight lines and detection of polygons in images were already proposed. Here we consider a similar model on machine for detection of circles in any image given that the image consists only of circles and nothing else. The algorithms for detection of straight lines [1], detection of polygons [2] may later be extended to encompass detection of simple primitive geometrical patterns. This in turn may be used to detect complex image patterns which may be decomposed into a set of such simple geometric patterns. Many image patterns e.g. man made machines, household articles fall in such category of images.

Detection of circles is an important task in many computer applications. These applications range from detection of tumors in medical images to detection of machine parts in industrial images. Any circle

has three parameters to represent; two co-ordinates of the center and radius. Hough transform is a robust technique that has long been used for such applications. The Hough transform maps points in an image to curves in a parameter space known as Hough space. Analytic and non-analytic curves in the image plane can be detected from the local maxima in the Hough space. Hough transform is computationally intensive. If we take n x n image, detection of circles in the image using Hough transform will take $O(n^4)$ time when gradient direction is not known. If gradient direction is known the complexity reduces to $O(n^3)$ time. Hence the detection of circles by Hough transform will depend on the size of the image. The best algorithm using Hough transform to find circles runs in $O(n^2)$ complexity. The algorithm we described here runs in $O(N)$ time to find all circles in the image, where $N$ is the number of circles in the image.

Chapter 2

# Problem Description & Assumptions

*Problem:*

Given that a binary image consists only of circular arcs and nothing else, we need to find coordinates of center and radii of all such circles.

*Assumptions:*

(1) Only the binary image is considered, hence any pixel in the image can have only two values 0 or 1.

(2) All circles in the image are continuous.

(3) All circles are of only one pixel width, i.e. some sort of pre processing (like thinning) was done on the image priorly

First let us see some general properties regarding circular arcs in an image.

(1) Circle follows 8 -way symmetry.

If one can draw quadrant 1 in the circle then remaining quadrants in the circle can be drawn symmetrically. If (x, y) is a pixel on the circle as in figure Fig. 1, the circle also contains the pixels (-x, y) (x, -y) (-x, -y) (y, x) (-y, x) (y, -x) (-y, -x) , assuming that the center of the circle is at the origin.

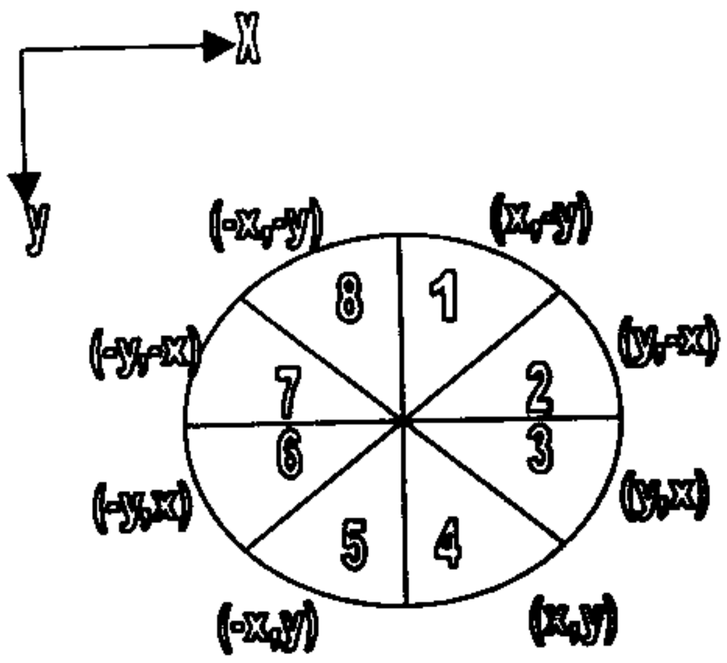



**Fig. 1** Symmetry Principle of circle

(2) If center coordinates are (x0, y0) and the radius is r, then the basic way to draw the circle is as follows:

For x = 1 to r do

y = ROUND( $\sqrt{}$( ($r^2 - (x-x0)^2$) ) )

/* ROUND will round the real number to nearest integer. */

Mark pixel (x+x0, y+y0).

Mark symmetrical pixels also.

End for

(3) If one calculates the curvature around an illuminated pixel on the circle this should match with that of another pixel. This property is used in the algorithm.

The image of a typical circle of radius 6 will be viewed as in the figure Fig. 2 assuming 1 as illuminated pixel and 0 as back ground.

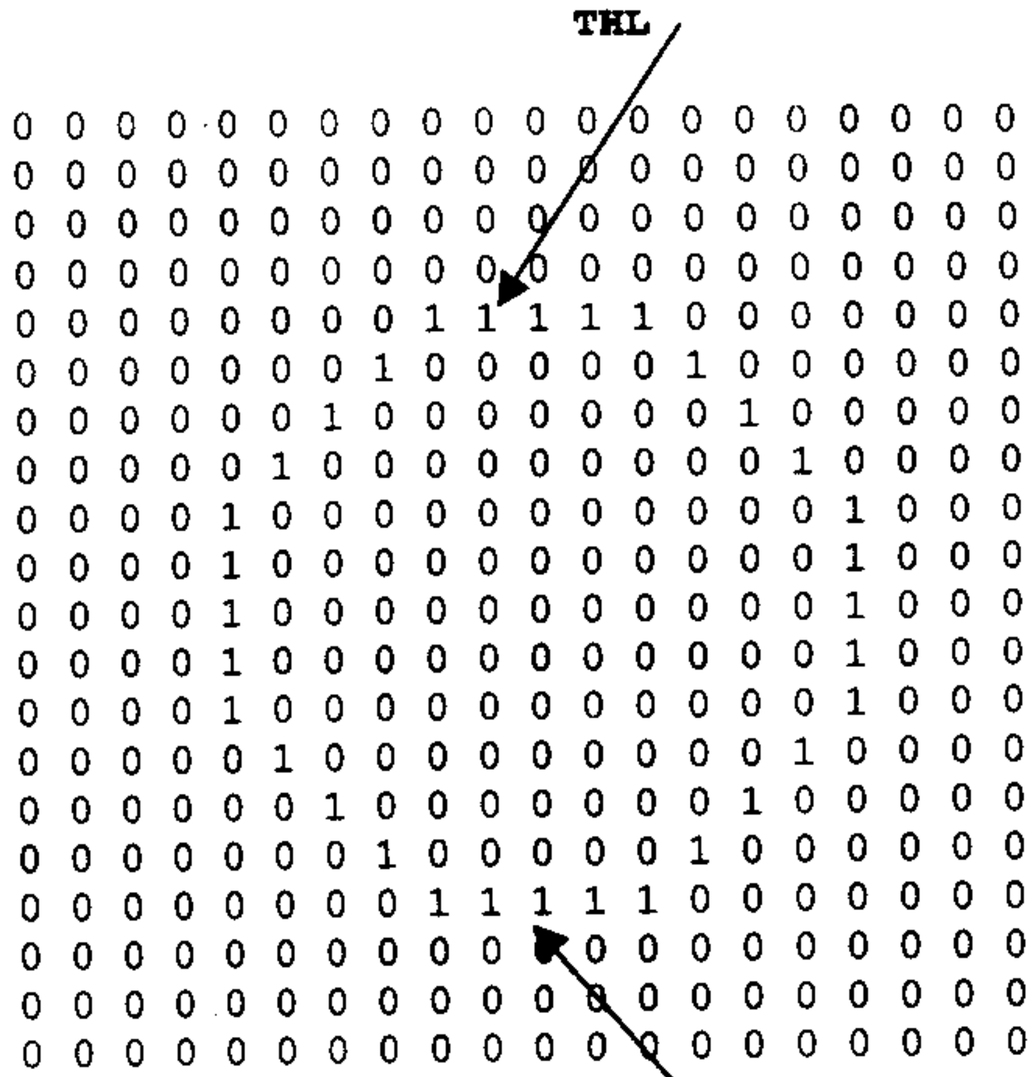



**Fig. 2** A Typical Circle of radius 6

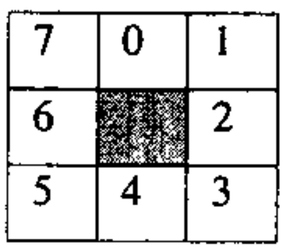
Any circle will contain a Top Horizontal Line (THL) and a Bottom Horizontal Line (BHL) as in figure Fig. 2. The length of THL (always odd) is same as BHL for a given circle. We define THL BHL as two types of horizontal lines at top and bottom of circle . Every circle got THL at the top and BHL at the bottom. Similarly we can define LVL (Left Vertical Line) and RVL(Right Vertical Line).

Chapter 3

# Categories of Illuminated Pixels

Any illuminated pixel information can be given by the coordinates of the pixel, the number of illuminated neighbors, the position of illuminated neighbors. The neighbors of any pixel on circle can assume only few positions depending on the quadrant of the circle in which it lies.

Let us consider the assignment for neighbors of a pixel in a 3 X 3 neighbors array of that pixel as follows.

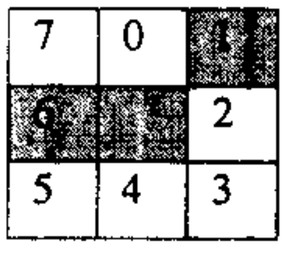



The illuminated pixels in a circle are classified into different types as follows.

Type A:

If neighbor pixels 1 and 6 only are illuminated then the pixel is categorized as point A.

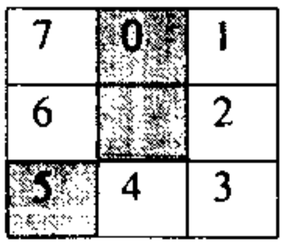
A pixel in 4 or 8 quadrants only can be of type A.




Point B:

If neighbor pixels 0 and 5 only are illuminated then the pixel is categorized as point B.

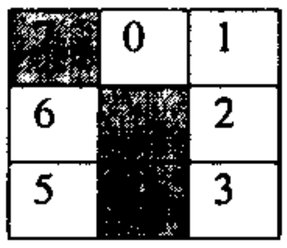
A pixel in 3 or 7 quadrants only can be of type B.



Type C:

If neighbor pixels 4 and 7 only are illuminated then the pixel is categorized as point C.

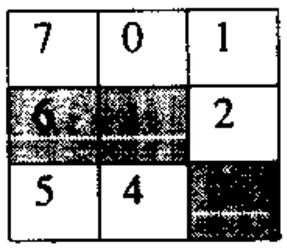
A pixel in 2 or 6 quadrants only can be of type C.




Type D:

If neighbor pixels 3 and 6 only are illuminated then the pixel is categorized as point D.

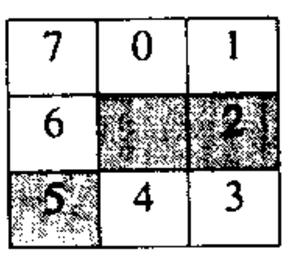
A pixel in 1 or 5 quadrants only can be of type D.




Type E:

If neighbor pixels 2 and 5 only are illuminated then the pixel is categorized as point E.

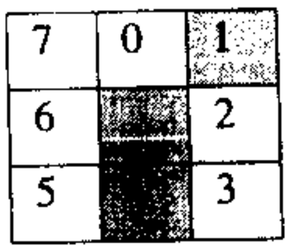
A pixel in 4 or 8 quadrants only can be of type E.




Type F:

If neighbor pixels 1 and 4 only are illuminated then the pixel is categorized as point F.

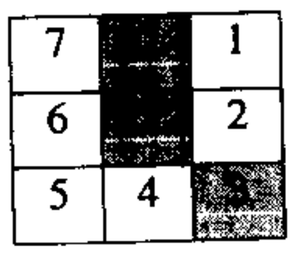
A pixel in 3 or 7 quadrants only can be of type F.




Type G:

If neighbor pixels 0 and 3 only are illuminated then the pixel is categorized as point G.

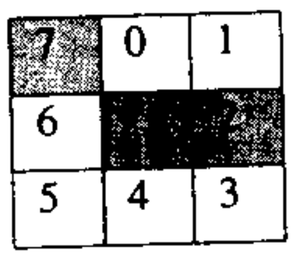
A pixel in 2 or 6 quadrants only can be of type G.




Type H:

If neighbor pixels 2 and 7 only are illuminated then the pixel is categorized as point H.

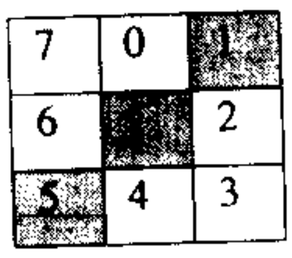
A pixel in 1 or 5 quadrants only can be of type H.




Type I:

If neighbor pixels 2 and 5 only are illuminated then the pixel is categorized as point I.

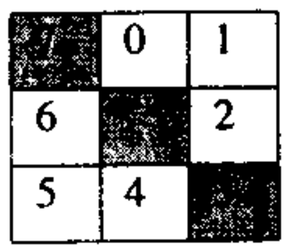
A pixel in 3 or 4 or 7 or 8 quadrants only can be of type I.



Type J:

If neighbor pixels 3 and 7 only are illuminated then the pixel is categorized as point J.

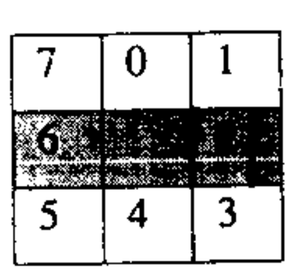
A pixel in 1 or 2 or 5 or 6 quadrants only can be of type J.




Type K:

If neighbor pixels 2 and 6 only are illuminated then the pixel is categorized as point K.

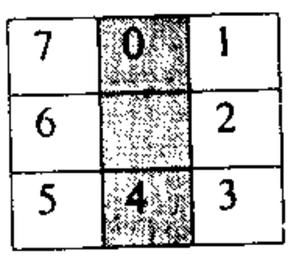
A pixel in 1 or 4 or 5 or 8quadrants only can be of type K.




Type L:

If neighbor pixels 0 and 4 only are illuminated then the pixel is categorized as point L.

A pixel in 2 or 3 or 6 or 7 quadrants only can be of type L.



Type *:

All other neighbor positions are combination of above types only. Such illuminated pixels are treated as type '*'. Actually these are intersection points of circles.

THL contains pixel of type E at left end and pixel of type D at right end. Pixels between those of type E and D are all of type K provided no intersections are there. Similarly BHL contains pixel of type H at left end and pixel of type A at right end. Pixels between those of type H and A are all of type K provided no intersections are there. The length of THL is one more to difference of x co-ordinates of D pixel and E pixel on it i.e. no of pixels on THL including E D pixels. Similarly length of BHL is one more to difference of x co-ordinates of H pixel and A pixel on it.

Chapter 4

# Curvature

Radius of curvature of a point on a curve is defined as

$$r = ((1 + (dy/dx)^2)^{3/2}) / (d^2y/dx^2) ,$$

where dy/dx is slope of curve at that point. $d^2y/dx^2$ is rate of change of slope. In an image slope is calculated as rate of change of coordinates of successive pixels on the curve. Similarly second derivative can be calculated.

Let some consecutive pixels on a curve be (x1, y1) (x2, y2) (x3, y3). Radius of curvature is calculated as

$(dy/dx_1) = (y2-y1)/(x2-x1)$ between pixels (x1,y1) and (x2,y2).

$(dy/dx_2) = (y3 - y2)/(x3 - x2)$ between pixels (x2,y2) and (x3,y3).

$d^2y/dx^2 = ( (dy/dx_1)-(dy/dx_2) )/(x1 - x3)$

Take average dy/dx and use these values in the formula above to calculate the radius of curvature at a pixel position. This is for three pixels. Similarly we can calculate the radius of curvature for Q number of successive pixels also.

If we calculate the local radius of curvatures at different points i.e. the radius of curvature for small number of pixels around the point on a circle, they should be equal. By taking some pixels near each THL and BHL the radius of curvature can be calculated. By comparing the radius of curvature a THL can be matched with a proper BHL.

In this algorithm we try to match given top horizontal lines with bottom horizontal lines by comparing local radii of curvatures. Once we know THL and BHL we can calculate center coordinates and radius of the circle exactly.

**Chapter 5**

# Architecture

The proposed architecture contains of $P$ X $P$ processor elements. Each processor element is denoted by PE$ij$. Each processor element processes a fixed zone of pixels in the image. We assume $n$ X $n$ binary image. This image is divided into $P$ X $P$ zones such that $P = n/m$, $m$ is an integer. We represent the zones as Z$ij$ $i, j = 0$ to $P - 1$. The processor element PE$ij$ works on Z$ij$ in the image. Each PE$ij$ will have some local memory (RAM). A Content Addressable memory CAM is used in accessing information speedily. CAM is divided into $P^2$ sub modules CAM$ij$ $i, j = 0$ to P - 1. Processor P$ij$ can write into sub module CAMi$j$ for storing some useful information about the pixel zone Z$ij$. But while searching CAM it was thought as $P$ X $P$ matrix.CAM can be searched in a row wise, column wise, 45 degrees slant angle wise also as in figure Fig. 3.

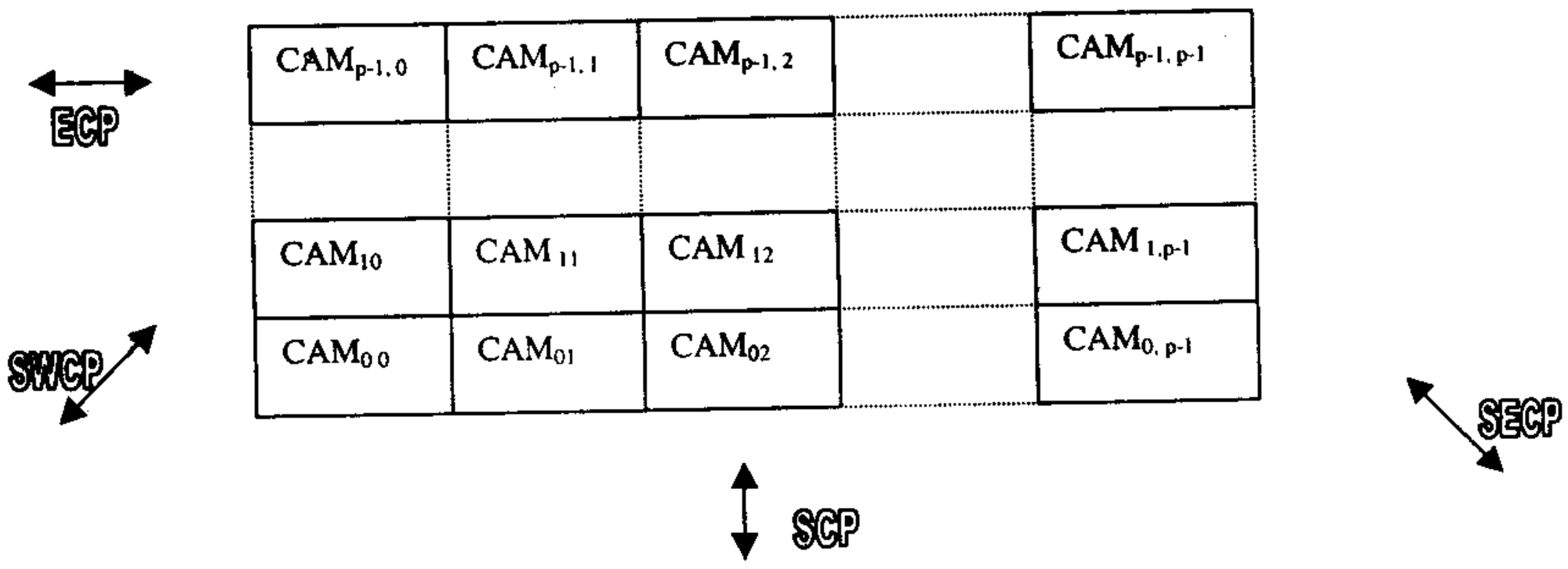



**Fig. 3** Cam Architecture

CAM can be used to increase the speed in searching. This is possible by using separate processors for searching. There will be east controlling processor (ECP), south controlling processor(SCP), south west controlling processors(SWCP), southeast controlling processor(SECP) as shown above.

$ECP_i$, $i$=0 to p – 1 operates on $i$th row

$SCP_i$, $i$=0 to p – 1 operates on $i$th column.

$SWCP_{ri}$, $ri$=0 to 2p –1 operates in direction south west (or north east). For example $SWCP_0$ works on $CAM_{p-1,0}$. $SWCP_1$ works on $CAM_{p-2,0}$ $CAM_{p-1,1}$ etc …

Similarly $SECP_{ri}$, $ri$=0 to 2p-1 operates on CAM modules in direction south east (or north west).

Each processor can give command to these controlling processors to search the CAM. A processor PE$ij$ can give command to number of controlling processors to search. This can be done by inter process communication within controlling processors. This inter process communication is effected through shared RAM working on EREW model.

Chapter 6

# Data structures

Each processor stores the information of all illuminated pixels in the zone it is working. Basically this is type information of the pixel. This info can be stored as an 8 bit word in the memory. For example a pixel as shown in figure fig. 4 below stores the information about neighbor as 10100001 where the 0th bit is rightmost one.

| | | |
|---|---|---|
| 1 | 1 | 0 |
| 0 | | 0 |
| 1 | 0 | 0 |

**Fig. 4**

From this 8 bit word the processor can decipher the type of the pixel. Also the information about illuminated pixels will be stored in CAM . The type, pixel position was stored in the CAM. Only the pixel types between A-H and * are stored in the CAM. Pixels I-K need not be stored in CAM.

| type | Pixel position |
|---|---|
| A | (x1,y1) |
| B | (x2,y2) |
| C | (x3,y3) |
| D | (x4,y4) |
| E | (x5,y5) |
| F | (x6,y6) |
| G | (x7,y7) |
| H | (x8,y8) |
| * | (x9,y9) |

**Fig. 5** Data Structure used in CAM

| Type | Left pixel | Right pixel | Radius of curvature |
|---|---|---|---|
| THL | (x1,y1) | (x2,y2) | ρ1 |
| BHL | (x3,y3) | (x4,y4) | ρ2 |

**Fig. 6** Data Structure used in CAM1

First each E pixel is matched with a possible right pixel of THP of a circle. This is done by checking, whether any D pixel exists successively on the same row as E. This information is stored in another content addressable memory CAM1. Similarly the possible BHL between H, A pixels is also stored in the CAM1. For each THP line and BHL line the local radius of curvature of the curve is calculated and stored in CAM1. Later Each THL is matched with BHL by comparing local radius of curvature. After matching is successful, the existence of circle is checked by illumination of intermediate close points on the circle.

Chapter 7

# Algorithm

The underlying idea of the algorithm is to pick THL (ED) line and BHL (HA) line and match them for a possible circle. Matching can be done easily by calculating local radius of curvature of ED HA portions. To find the local curvature of a pixel, the adjacent pixel information can be used. CAM is used to get possible adjacent pixel information. Controlling processors are used for this purpose. Still if there are several opposite horizontal lines to be considered, each of them is compared by checking the global radius of curvature. Later the illumination of some close and intermediate points on the circle is also checked for possible circle.

## *7.1 Algorithm*

**Step 1:**

**1.1** The processing element PE*ij* conveys the information about the positions of the illuminated pixels lying on the border of zone Z*ij* to the respective neighbor processing elements, whenever they exist.

**1.2** P*ij* checks the types of illuminated pixels in its zone. If the pixel type is one of A to H or *, then the position and type information is entered into CAM submodule corresponding to that PE.

**Step 2:**

**2.1** For every pixel of type E (x1,y1) found, P*ij* will search for successive pixel type D in the same row with no other types of pixels in between by issuing commands to CAM row processor.

Pixel position is (x1, y1): ECP finds all pixels (x, y1) of type D; the pixel (x2,y1) is chosen from such (x,y1) where x2 is the minimum of all x and also x2 > x1.

Similarly check for nearest pixel of different types.

If pixel of type D is the nearest pixel than all other types of successive pixels, E is associated with D. The matching ED line (type THL) is noted in CAM1. In it's local memory also the line info is noted down.

**2.2** Similarly for every pixel of type H (x1,y1) found, PE*ij* will search for successive pixel type A in the same row with no other types of pixels in between by issuing commands to CAM row processor.

Pixel position is (x1, y1): ECP finds all pixels (x,y1) of type A; the pixel (x2,y1) is chosen from such (x, y1) where x2 is the minimum and also x2 > x1.

Similarly check for nearest pixel of different types.

If pixel of type A is the nearest pixel than all other types of successive pixels, H is associated with A. The matching HA line (type BHL) is noted in CAM1. In it's local memory also the line info is noted down.

**Step 3:**

Now PE*ij* will calculate local curvature for each THL line or BHL line by finding Q maximum successive pixels from the right end pixel (i.e. D for THL, A for BHL) by issuing command to the CAM controlling processors as follows.

**3.1** When line is of type THL and right end pixel (of type D) is (x1, y1)

When (x1, y1) is of type D: command to SECP to find nearest pixel of type C or H (x, y) for x > x1 y > y1.

When (x1, y1) is of type C: command to ECP to find nearest pixel of type B or G (x1, y) for y > y1

When (x1, y1) is of type H: command to ECP to find nearest pixel of type D (x, y1) for x>x1

When (x1, y1) is of type G: command to SECP to find nearest pixel of type C (x,y) for x>x1 ,y > y1.

If the successive pixel got does not fall in above category then the ED pixels does not belong to top horizontal line of any circle. The line is entered as "invalid". Calculation is aborted. Above process is done till Q number of pixels were found out Or pixel of type B is found out whichever is first one.

**3.2** Similar to above PE*ij* will calculate local curvature for each HA line as follows.

When (x1,y1) is of type A: command to SWCP to find nearest pixel is of type E or B (x, y) for x > x1 y < y1.

When (x1, y1) is of type B: command to SCP to find nearest pixel is of type F or C (x1, y) for y < y1.

When (x1, y1) is of type E: command to ECP nearest pixel of type A (x, y1) for x>x1.

When (x1,y1) is of type F: command to SWCP nearest pixel is of type B (x, y) for x > x1, y < y1 .

If the successive pixel got does not fall in above category then the HA pixels does not belong to bottom horizontal line of any circle. The line is entered as "invalid". Calculation is aborted. Above process is done till Q number of pixels were found out Or pixel of type C is found out whichever is first one.

**3.3** Mark all THL BHL which are not marked "invalid" as "unmatched"

**Step 4:** For every unmatched THL or BHL lines found, processor P*ij* will try to match a possible opposite horizontal line (i.e. other type of line) of same length and same curvature as follows:

**4.1** P*ij* issues a command to all RCP's for opposite horizontal line of same curvature in CAM1. If no such portion was found out then P*ij* marks the portion as "match_not_yet_found"

**4.2** For each such opposite horizontal lines, find center co ordinates(x0,y0) and radius r.

Interpolate say T no of points on the circle and find whether they are illuminated. If they are illuminated then mark THL and BHL as "matched" else "match_not_yet_found".

**Step 5:** Now remaining circles to be detected have at least one pixel of type * on THL or BHL lines.

Each PE*ij* issues command to ECP

**5.1** where the pixel is of type E: Position is (x1,y1) take all * pixels (x,y1) and the nearest D point(x2,y2).All * pixels (x3,y1) x3 > x1and x3 < x2 are taken as possible D points and entered into CAM1.

**5.2** where the pixel is of type H: Position is (x1,y1) take all * pixels (x,y1) and the nearest A point(x2,y2). All * pixels (x3,y1) x3 > x1 and x3 < x2 are taken as possible A points and entered into CAM1.

**5.3** where the pixel is of type *: Position is (x1, y1) take all * pixels (x,y1) and the nearest D or A points. All * pixels before D or A are considered as right side of THL as well BHL and are entered into CAM1. Pixels D A also are entered into CAM1 as right pixels of THL and BHL.

**Step 6:** PE*ij* will try to find curvature of THL BHL entered newly into curvature.

The lines of type THL and BHL which are not marked "invalid" are marked "unmatched".

**6.1** All "match_not_yet_found" are marked as "unmatched" and the matching process is repeated again as above.

### *7.2 Complexity*

Step 1 can be done in constant time. Step 2 can be done in $O(logN)$ time since sorting need to be done to get the minimum x coordinate pixel. On any horizontal line of image, the maximum number of pixels of same type is 2N. Step 3 can be done in $O(QlogN)$ time. Step 3.1 or 3.2 can be done in $O(logN)$ time since sorting need to be done on $O(N)$ pixels. Step 3.1 or 3.2 is repeated maximum Q number of times where Q is a constant.

Step 4 can be done again in $O(N)$ since maximum number of horizontal lines of opposite type of same radius of curvature to be compared is N. In simple case CAM1 contains THL or BHL of all circles. In other cases all THL BHL are not written in CAM1. Step 5 again takes $O(logN)$ time since sorting is needed. Step 5.1 5.2 5.3 takes $O(N)$ time. Number of entries in CAM1 in step 5 depend on number of intersections pairs on all horizontal line. The maximum number of intersection pairs that occur on a single horizontal line is 2N. Step 6 requires $O(N)$ time.

Combining all these the complexity of the algorithm is $O(N)$.

Chapter 8

# Conclusions

Since THL BHL are considered, only continuous circles without any breaks in between can be detected. Also the image is assumed to be of noise free. Therefore before applying this algorithm the image should be preprocessed to remove noise.

A time optimal algorithm for detection of *N* circles was described assuming image contains only continuous circles. The execution time of the algorithm is insensitive to center co ordinates and radii of circles. The algorithm has been developed on a model involving content addressable memory(CAM) modules. The algorithm may be extended freely to find curves like ellipse etc.

# References

[1] Amitava Sen, Mallika De and Bhabani P. Sinha, *Time-Optimal Algorithm for identification of Straight Lines in Images*: Technical Report No:CCSD/98/BPS/01, ISI, 1998.

[2] Arijit Laha, *Parallel Architecture and Algorithms for Fast Detection of Basis Polygons*: M.Tech Computer Science Dissertation Series, ISI, 1996

[3] R.C. Gonzalez and R.E. Woods, *Digital Image Processing*. New York : Addison-Wesley, 1992.

# Appendix

The C ++ implementation of the algorithm is given below. The program contains three files

Pro.cpp

Image.h

Cam.h

The algorithm was developed in the two functions find_cir1() and find_cir2() of class Image.

File cam.h

```
#define ERR '#'
enum BOOL {false,true};
BOOL is_brk(char c);

class Circle
{
      int x0,y0;    /* Center in terms of radius units */
      float rad;
public:
      Circle()
      {
            x0 = y0 = -1;
            rad = -1;
      }
      Circle(int x,int y,float r)
      {
            x0 = x;
            y0 = y;
            rad = r;
      }
      Circle(int x,int y,int r)
      {
            x0 = x;
            y0 = y;
            rad = r;
      }
      gety(int x)
      {
      return (sqrt(rad*rad - x*x)+0.5);
      }
};
```

```
class Cam      // Size = max index of no of elements
{
      int size;
      int array[4*SIZE][4];
      public:

            int get_0brkpt(int i);
            int get_1brkpt(int i);     // For different directions
            int get_2brkpt(int i);     // These fns are used
            int get_3brkpt(int i);      // to get nearest point
            int get_4brkpt(int i);
            int get_5brkpt(int i);
            int get_6brkpt(int i);
            int get_7brkpt(int i);

            Cam();
            int get_size();

            char get_char(int x,int y);
            // Gets array[2] given array[0] array[1]
            void read_cam(int x,int y,int c,int mark);
            // Reads contents into Cam
            void print(int i);
            // Prints i th contents of cam
            void put_cam(int k,int&x,int&y,int&z);
            // Puts k th values into x y z
            int get3(int i);
            //  gets char given index i
            int index(int x,int y);
            // Gives the index
            void mark(int x,int y);
            int get_unmarkpt();
            // Gets some un marked pt
};

void Cam::mark(int x,int y)
{
      int k = index(x,y);
      array[k][3] = 1;
      return;
}

int Cam::index(int x,int y)
{
      int i,j;
      for(i = 0;i <= size;i ++)
      {
            if((array[i][0] == x)&&(array[i][1] == y))
                  return i;
      }
      return -1;
}
```

```
Cam::Cam()
{
      size = -1;
}
int Cam::get_size()
{
      return size;
}
char Cam::get_char(int x,int y)
// gets char from CAM
{
int i,j;
for(i = 0;i <= size; i ++)
{
      if((array[i][0]==x)&&(array[i][1]==y))
            return (array[i][2]);
}
return ERR;

}
void Cam::read_cam(int x,int y,int c,int mark)
{
      size ++;
      int i = size;
      array[i][0] = x;
      array[i][1] = y;
      array[i][2] = c;
      array[i][3] = mark;
}
void Cam::print(int i)
{
if(i <= size)
printf("%d %d %c    ",array[i][0],array[i][1],array[i][2]);
else
printf("CAM don't have such value\n");
}
void Cam::put_cam(int k,int&x,int&y,int&z)
{
      x = array[k][0];
      y = array[k][1];
      z = array[k][2];
}
int Cam::get3(int i)
{
      return array[i][3];
}

int Cam::get_unmarkpt()
{
      int i;
      for(i = 0; i <= size; i ++)
      {
            char c = array[i][2];
            int d = array[i][3];
if((d == 0)&&(c != 'I')&&(c != 'J')&&(c!= 'K')&&(c !='L'))
{
```

```
                return i;
        }
                }
        }

int Cam::get_0brkpt(int i)
{
        int x = array[i][0];
        int y = array[i][1];
        char c = array[i][2];
        int x1,y1;
        char c1;
        x ++;
        y ++;
        int j = i;
        while(j < size)
        {
                c1 = get_char(x,y);
                if((c1 != ERR)&&(is_brk(c1) == true))
                {
                        return index(x,y);
                }
                j ++;
                x ++;
                y ++;
        }
        return -1;
}
int Cam::get_1brkpt(int i)
{
int x = array[i][0];
int y = array[i][1];
char c = array[i][2];
int x1,y1;
char c1;
x ++;
int j = i;
while(j < size)
{
        c1 = get_char(x,y);
        if((c1 != ERR)&&(is_brk(c1) == true))
        {
                return index(x,y);
        }
        x ++;
        j ++;
}
return -1;
}
int Cam::get_2brkpt(int i)
{
        int x = array[i][0];
        int y = array[i][1];
        char c = array[i][2];
        int x1,y1;
        int c1;
        x ++;
```

```
y --;
int j = i;
while(j  >= 0)
{
      c1 = get_char(x,y);
      if((c1 != ERR)&&(is_brk(c1) == true))
      {
            return index(x,y);
      }
      x ++;
      y --;
            j --;
      }
      return -1;
}
int Cam::get_3brkpt(int i)
{
      int x = array[i][0];
      int y = array[i][1];
      char c = array[i][2];
      int x1,y1;
      int c1;
      y --;
      int j = i;
      while(j >= 0)
      {
            c1 = get_char(x,y);
            if((c1 != ERR)&&(is_brk(c1) == true))
            {
                  return index(x,y);
            }
            y --;
            j --;
      }
      return -1;
}
int Cam::get_4brkpt(int i)
{
      int x = array[i][0];
      int y = array[i][1];
      char c = array[i][2];
      int x1,y1;
      int c1;
      x --;
      y --;
      int j = i;
      while(j >= 0)
      {
      c1 = get_char(x,y);
      if((c1 != ERR)&&(is_brk(c1) == true))
      {
            return index(x,y);
      }
      x --;
      y --;
      j --;
}
```

```
        return -1;
}
int Cam::get_5brkpt(int i)
{
        int x = array[i][0];
        int y = array[i][1];
        char c = array[i][2];
        int x1,y1;
        int c1;
        x --;
        int j = i;
        while((j >=0))
        {
                c1 = get_char(x,y);
                if((c1 != ERR)&&(is_brk(c1) == true))
                {
                        return index(x,y);
                }
                x --;
                j --;
        }
        return -1;
}
int Cam::get_6brkpt(int i)
{
        int x = array[i][0];
        int y = array[i][1];
        char c = array[i][2];
        int x1,y1;
        int c1;
        x --;
        y ++;
        int j = i;
        while(j < size)
        {
                c1 = get_char(x,y);
                if((c1 != ERR)&&(is_brk(c1) == true))
                {
                        return index(x,y);
                }
                x --;
                y ++;
                j ++;
        }
        return -1;
}
int Cam::get_7brkpt(int i)
{
        int x = array[i][0];
        int y = array[i][1];
        char c = array[i][2];
        int x1,y1;
        int c1;
        int j = i;
        y ++;
        while(j < size)
        {
```

```
                c1 = get_char(x,y);
                if((c1 != ERR)&&(is_brk(c1) == true))
                {
                        return index(x,y);
                }
                y ++;
                j ++;
        }
        return -1;
}
BOOL is_brk(char c)
{
        if((c == 'A')||(c == 'B')||(c == 'C')||(c == 'D')||(c == 'E')||
                (c == 'F')||(c == 'G')||(c == 'H')||(c == '*'))
        {
                return true;
        }
                else
                        return false;
}
```

File image.h

```
#include <d:\sudha\finproj\cam.h>
#define brk_no = 3;

Cam cam;
BOOL equal(int* pt,int* p);
// Checks whether p is any of the types A - H

struct Cam1 // cam1 in the algorithm
{
int pq1[SIZE][3][2];
int pq2[SIZE][3][2];
int pq3[SIZE][3][2];
int pq4[SIZE][3][2];
int pr1[SIZE];
int pr2[SIZE];
int pr3[SIZE];
int pr4[SIZE];
int NUMBER1;
int NUMBER2;
int NUMBER3;
int NUMBER4;

Cam1()
{
NUMBER1 = -1;
NUMBER2 = -1;
NUMBER3 = -1;
NUMBER4 = -1;
}
```

```
} cam1;

class Image1  // This is used as an accessory to Image below..
{
      int size;
      char array[SIZE][SIZE];
      public:
      Image1(int s)
      {
      size = s;
      }
      char get_value(int x,int y)
      {
            return array[x][y];
      }
      void read_value(int x,int y,char c)
      {
            array[x][y] = c;
      }
      void reset();
      void draw_image();
};
void Image1::reset()
{
      int i,j;
      for(i = 0;i < SIZE; i ++)
      for(j = 0;j < SIZE;j ++)
      {
            array[i][j] = '0';
      }
}

void Image1::draw_image()
{
      for(int i = 0;i < size;i ++)
      {
      for(int j =0; j < size; j ++)
      {
            printf(" %c",array[i][j]);
      }
            printf("\n");
      }
}

class Image
// This contains all parts to process Image given.
{
      int size;
int array[SIZE][SIZE];
public:
      Image(int s)
      {
            size = s;
      }
      int get_size()
      {
            return size;
```

```
}
void reset();
void draw_image();
void put_circle();

void fin1_cir(Image1* brk);
void fin2_cir(Image1* brk);
void find_circle();
// Finds Circle by algo

                BOOL chk_cir1(int x,int y,int r);
                BOOL chk_cir2(int x,int y,int r);

                int next_brkpt(int in);
                BOOL find_der(int i,int j);
                void find_break(Image1* brk);

};
void Image::reset()
{
      int i,j;
      for(i = 0;i < SIZE; i ++)
      for(j = 0;j < SIZE;j ++)
      {
            array[i][j] = 0;
      }
}
void Image::draw_image()
// Draws Image
{
      for(int i = 0;i < size;i ++)
      {
      for(int j =0; j < size; j ++)
      {
            printf(" %d",array[i][j]);
      }
            printf("\n");
      }
}


void Image::put_circle()
// Puts Circle in the image of size
{
      int x0,y0;
      printf("Enter Center:");
      scanf("%d",&x0);
      scanf("%d",&y0);
      printf("Center: %d %d ",x0,y0);
      int rad;
      printf("Enter rad:");
      scanf("%d",&rad);
      printf("rad: %d\n",rad);

      Circle c(x0,y0,rad);
      for(int i = 0; i < rad; i ++)
      {
```

```
int x = i;
int y = c.gety(i);
int x1 = x0+x;
int y1 = y0+y;
if((x1 > 0)&&(y1 > 0)&&(x1 < size)&&(y1 < size))
{
      array[x1][y1] = 1;
}
else
{
      printf("Error in input\n");
      exit(0);
}
x1 = x0 -x ;
y1 = y0 +y;
if((x1 > 0)&&(y1 > 0)&&(x1 < size)&&(y1 < size))
{
      array[x1][y1] = 1;
}
else
{
      printf("Error in input\n");
      exit(0);
}

x1 = x0 +x ;
y1 = y0 -y;
if((x1 > 0)&&(y1 > 0)&&(x1 < size)&&(y1 < size))
{
      array[x1][y1] = 1;
}
else
{
      printf("Error in input\n");
      exit(0);
}

x1 = x0 -x ;
y1 = y0 -y;
if((x1 > 0)&&(y1 > 0)&&(x1 < size)&&(y1 < size))
{
      array[x1][y1] = 1;
}
else
{
      printf("Error in input\n");
      exit(0);
}

x1 = y0 +y ;
y1 = x0 +x;
if((x1 > 0)&&(y1 > 0)&&(x1 < size)&&(y1 < size))
{
      array[x1][y1] = 1;
}
else
{
```

```
        printf("Error in input\n");
        exit(0);
}

x1 = y0 -y ;
y1 = x0 +x;
if((x1 > 0)&&(y1 > 0)&&(x1 < size)&&(y1 < size))
{
        array[x1][y1] = 1;
}

else
{
        printf("Error in input\n");
        exit(0);
}

x1 = y0 +y ;
y1 = x0 -x;
if((x1 > 0)&&(y1 > 0)&&(x1 < size)&&(y1 < size))
{
        array[x1][y1] = 1;
}
else
{
        printf("Error in input\n");
        exit(0);
}

                x1 = y0 -y ;
                y1 = x0 -x;
                if((x1 > 0)&&(y1 > 0)&&(x1 < size)&&(y1 < size))
                {
                        array[x1][y1] = 1;
                }
                else
                {
                        printf("Error in input\n");
                        exit(0);
                }

        }
}
void Image::find_circle()
// Finds Circle by algo
{
        int pt[NO][9];
        pt[0][0] = 0;
        pt[0][1] = 0;
        pt[0][2] = 1;
        pt[0][3] = 1;
        pt[0][4] = 1;
        pt[0][5] = 0;
        pt[0][6] = 0;
        pt[0][7] = 0;
        pt[0][8] = 0;
```

```
pt[1][0] = 0;
pt[1][1] = 1;
pt[1][2] = 0;
pt[1][3] = 0;
pt[1][4] = 1;
pt[1][5] = 0;
pt[1][6] = 1;
pt[1][7] = 0;
pt[1][8] = 0;

pt[2][0] = 1;
pt[2][1] = 0;
pt[2][2] = 0;
pt[2][3] = 0;
pt[2][4] = 1;
pt[2][5] = 0;
pt[2][6] = 0;
pt[2][7] = 1;
pt[2][8] = 0;

pt[3][0] = 0;
pt[3][1] = 0;
pt[3][2] = 0;
pt[3][3] = 1;
pt[3][4] = 1;
pt[3][5] = 0;
pt[3][6] = 0;
pt[3][7] = 0;
pt[3][8] = 1;

pt[4][0] = 0;
pt[4][1] = 0;
pt[4][2] = 0;
pt[4][3] = 0;
pt[4][4] = 1;
pt[4][5] = 1;
pt[4][6] = 1;
pt[4][7] = 0;
pt[4][8] = 0;

pt[5][0] = 0;
pt[5][1] = 0;
pt[5][2] = 1;
pt[5][3] = 0;
pt[5][4] = 1;
pt[5][5] = 0;
pt[5][6] = 0;
pt[5][7] = 1;
pt[5][8] = 0;

pt[6][0] = 0;
pt[6][1] = 1;
pt[6][2] = 0;
pt[6][3] = 0;
pt[6][4] = 1;
pt[6][5] = 0;
pt[6][6] = 0;
```

```
pt[6][7] = 0;
pt[6][8] = 1;

pt[7][0] = 1;
pt[7][1] = 0;
pt[7][2] = 0;
pt[7][3] = 0;
pt[7][4] = 1;
pt[7][5] = 1;
pt[7][6] = 0;
pt[7][7] = 0;
pt[7][8] = 0;

pt[8][0] = 0;
pt[8][1] = 0;
pt[8][2] = 1;
pt[8][3] = 0;
pt[8][4] = 1;
pt[8][5] = 0;
pt[8][6] = 1;
pt[8][7] = 0;
pt[8][8] = 0;

pt[9][0] = 1;
pt[9][1] = 0;
pt[9][2] = 0;
pt[9][3] = 0;
pt[9][4] = 1;
pt[9][5] = 0;
pt[9][6] = 0;
pt[9][7] = 0;
pt[9][8] = 1;

pt[10][0] = 0;
pt[10][1] = 0;
pt[10][2] = 0;
pt[10][3] = 1;
pt[10][4] = 1;
pt[10][5] = 1;
pt[10][6] = 0;
pt[10][7] = 0;
pt[10][8] = 0;

pt[11][0] = 0;
pt[11][1] = 1;
pt[11][2] = 0;
pt[11][3] = 0;
pt[11][4] = 1;
pt[11][5] = 0;
pt[11][6] = 0;
pt[11][7] = 1;
pt[11][8] = 0;


int sz = get_size();
Image1* brk = new Image1(sz);
brk->reset();
```

```
        printf("Circle finding started\n");
        find_break(brk);

        char cam1[SIZE];
        for(int i = 0; i < size;i ++)
        for(int j = 0;j < size;j ++)
// Stores the pts in CAM.
        {
                char c = brk->get_value(i,j);
                if(c != '0')
                {
                        cam.read_cam(j,i,c,0);
                }
        } // double for loop

        fin1_cir(brk);
        fin2_cir(brk);
}
void Image::fin1_cir(Image1* brk)
// Finds Circle by algo
{
        for(int i = 0;i < size; i ++)
        for(int j = 0;j < size; j ++)
        {
                char c;
                int pp[3];
        // 3 no of continous brek pt s are taken ..
                pp[0] = pp[1] = pp[2] = -1;
                // 3 -1 = 2 first comenent of array
                int p,q,r;
                int u,v,w;
                char c1;
                int k,k1;
                int temp;
if((c = cam.get_char(i,j)) != ERR)
switch(c)
{
                case('E'):
                k = cam.index(i,j);
                k1 = next_brkpt(k);
                cam.put_cam(k1,u,v,w);
if(w != 'D')
                break;
                cam1.NUMBER1 ++;
                temp = 1;
                pp[0] = k;
                cam1.pq1[cam1.NUMBER1][0][0] = i;
                cam1.pq1[cam1.NUMBER1][0][1] = j;

while(temp >= 0)
{
cam.put_cam(k,u,v,w);
c = w;
k1 = next_brkpt(k);

cam.put_cam(k1,p,q,r);
c1 = r;
```

```
cam1.pq1[cam1.NUMBER1][2-temp][0] = p-i;
cam1.pq1[cam1.NUMBER1][2-temp][1] = q-j;
if((c == 'C')&&(r == 'B'))
      break;
pp[2 - temp] = k1;
k = k1;
temp --;
}
      break;
case('H'):
k = cam.index(i,j);
k1 = next_brkpt(k);
cam.put_cam(k1,u,v,w);
if(w != 'A')
      break;
cam1.NUMBER2++;
temp = 1;
pp[0] = k;
cam1.pq2[cam1.NUMBER2][0][0] = i;
cam1.pq2[cam1.NUMBER2][0][1] = j;
while(temp >= 0)
{
cam.put_cam(k,u,v,w);

c = w;
k1 = next_brkpt(k);

cam.put_cam(k1,p,q,r);
c1 = r;
cam1.pq2[cam1.NUMBER2][2-temp][0] = p-i;
cam1.pq2[cam1.NUMBER2][2-temp][1] = q-j;
if((c == 'B')&&(r == 'C'))
      break;
pp[2 - temp] = k1;
k = k1;
temp --;
}
      break;
default:
      break;
};

}// for

// Comparision starts here to ascertain circles ..
for(i = 0;i < SIZE; i ++)
{
      cam1.pr1[i] = 0;
      cam1.pr2[i] = 0;
      cam1.pr3[i] = 0;
      cam1.pr4[i] = 0;
}

for(i = 0;i <= cam1.NUMBER1; i ++)
for(j = 0;j <= cam1.NUMBER2; j ++)
```

```
        {
        BOOL BL = false;
        for(int k = 1;(k < 3&&BL == false); k ++)
        {

        if((cam1.pq1[i][k][0]==cam1.pq2[j][k][0])&&(cam1.pq1[i][k][1] ==-
cam1.pq2[j][k][1]))
                {
                        BL = false;
                }
                else
                {
                        BL = true;
                }
}
                if(BL == false)
                        // Circle found out ..
                {
                        int x1 = cam1.pq1[i][0][0];
                        int y1 = cam1.pq1[i][0][1];
                        int x2 = x1+cam1.pq1[i][1][0];
                        int y2 = y1+cam1.pq1[i][1][1];
                        int x3 = cam1.pq2[j][0][0];
                        int y3 = cam1.pq2[j][0][1];
                        int x4 = x3+cam1.pq2[j][1][0];
                        int y4 = y3+cam1.pq2[j][1][1];

                        int w = (x2 - x1)/2;
                        int h = (y3 - y1)/2;
                        int cx = x1+w;
                        int cy = y1+h;
                        int r = sqrt((cx-x1)*(cx-x1)+(cy-y1)*(cy-y1));
                        if(find_der(i,j) == true)
                        if((chk_cir1(cx,cy,r) == true)&&(r > 1))
                        {
                        cam1.pr1[i] = 1;
                        cam1.pr2[j] = 1;
                        printf("Circle: %d %d %d \n",cx,cy,r);
                        getch();
                        }

// Mark the pixels that they were found out ..
cam.mark(x1,y1);
cam.mark(x2,y2);
cam.mark(x3,y3);
cam.mark(x4,y4);
                }
        }

}
void Image::fin2_cir(Image1* brk)
{
// Find remaining circles in the image by algo

        for(int i = 0;i < size; i ++)
        for(int j = 0;j < size; j ++)
        {
```

```
        char c;
        int pp[3];
// 3 no of continous break pts are taken ..
        pp[0] = pp[1] = pp[2] = -1;
        // 3 -1 = 2 first comenent of array
                int p,q,r;
                int u,v,w;
                char c1;
                int k,k1;
                int temp;
if((c = cam.get_char(i,j)) != ERR)
switch(c)
{
        case('E'):
        k = cam.index(i,j);
int PP = i;
while(PP < size)
{
        BOOL lk = false;
        PP ++;
         char s = brk->get_value(j,PP);

if((s != '*')&&(s != 'D')&&(is_brk(s)))
{
        break;
}
if(((s == 'D')&&(lk == true))||(s == '*'))
{
if(s == '*')
                lk = true;
k1 = cam.index(PP,j);
cam.put_cam(k1,u,v,w);
cam1.NUMBER3 ++;
temp = 0;
pp[0] = k;
cam1.pq3[cam1.NUMBER3][0][0] = i;
cam1.pq3[cam1.NUMBER3][0][1] = j;

cam.put_cam(k,u,v,w);
c = w;
cam.put_cam(k1,p,q,r);
cam1.pq3[cam1.NUMBER3][1][0] = p-i;
cam1.pq3[cam1.NUMBER3][1][1] = q-j;
int k2 = next_brkpt(k1);
if(k2 != -1)
{
                int p,q,r;
                cam.put_cam(k2,p,q,r);
                cam1.pq3[cam1.NUMBER3][2][0] = p - i;
                cam1.pq3[cam1.NUMBER3][2][1] = q - j;
        }
        if(s == 'D')
                break;
        } //if

} // while loop
                break;
```

```
        case('H'):

        k = cam.index(i,j);
PP = i;
while(PP < size)
{
        BOOL lk = false;
        PP ++;
         char s = brk->get_value(j,PP);

if((s != 'A')&&(s != '*')&&(is_brk(s)))
{
        break;
}
if(((s == 'A')&&(lk == true))||(s == '*'))
{
        if(s == '*')
                        lk = true;

        k1 = cam.index(PP,j);
        cam.put_cam(k1,u,v,w);
        cam1.NUMBER4 ++;
        temp = 0;
        pp[0] = k;
        cam1.pq4[cam1.NUMBER4][0][0] = i;
        cam1.pq4[cam1.NUMBER4][0][1] = j;

        cam.put_cam(k,u,v,w);
                c = w;
                cam.put_cam(k1,p,q,r);
                cam1.pq4[cam1.NUMBER4][1][0] = p-i;
                cam1.pq4[cam1.NUMBER4][1][1] = q-j;
                if(s == 'A')
                        break;
        } //if

} // while loop


                        break;

                case '*':


        k = cam.index(i,j);
        PP = i;
while(PP < size)
{
        BOOL lk = false;
        PP ++;
         char s = brk->get_value(j,PP);

if((s != 'D')&&(s != 'A')&&(s != '*')&&(is_brk(s)))
{
        break;
}
if((s == 'D')||(s == '*'))
```

```
{
        k1 = cam.index(PP,j);
        cam.put_cam(k1,u,v,w);
        cam1.NUMBER3 ++;
        temp = 0;
        pp[0] = k;
        cam1.pq3[cam1.NUMBER3][0][0] = i;
        cam1.pq3[cam1.NUMBER3][0][1] = j;

        cam.put_cam(k,u,v,w);
        c = w;
        cam.put_cam(k1,p,q,r);
        cam1.pq3[cam1.NUMBER3][1][0] = p-i;
        cam1.pq3[cam1.NUMBER3][1][1] = q-j;


if(s == 'D')
        break;
} //if
if((s == 'A')||(s == '*'))
{


        k1 = cam.index(PP,j);
        cam.put_cam(k1,u,v,w);
        cam1.NUMBER4 ++;
        temp = 0;
        pp[0] = k;
        cam1.pq4[cam1.NUMBER4][0][0] = i;
        cam1.pq4[cam1.NUMBER4][0][1] = j;
                cam.put_cam(k,u,v,w);
                c = w;
                cam.put_cam(k1,p,q,r);
                cam1.pq4[cam1.NUMBER4][1][0] = p-i;
                cam1.pq4[cam1.NUMBER4][1][1] = q-j;
        if(s == 'A')
                break;
        } //if
        } // while loop

                break;
                default:
                        break;
        }; // switch

}// for

// Comparision starts here ........

for(i = 0;i <= cam1.NUMBER3; i ++)
for(j = 0;j <= cam1.NUMBER4; j ++)
if((cam1.pr3[i] == 0)&&(cam1.pr4[j] == 0))
{
BOOL BL = false;
for(int k = 1;(k < 2&&BL == false); k ++)
```

```
        {

        if((cam1.pq3[i][k][0]==cam1.pq4[j][k][0])&&(cam1.pq3[i][k][1] ==
-cam1.pq4[j][k][1]))
                if(cam1.pq3[i][k][1] != 1)
                {
                        BL = false;
                }
                else
                {
                        BL = true;
                }
        }
                if(BL == false)
                        // Circle found out ..
                {
                        int x1 = cam1.pq3[i][0][0];
                        int y1 = cam1.pq3[i][0][1];
                        int x2 = x1+cam1.pq3[i][1][0];
                        int y2 = y1+cam1.pq3[i][1][1];
                        int x3 = cam1.pq4[j][0][0];
                        int y3 = cam1.pq4[j][0][1];
                        int x4 = x3+cam1.pq4[j][1][0];
                        int y4 = y3+cam1.pq4[j][1][1];

                        int w = (x2 - x1)/2;
                        int h = (y3 - y1)/2;
                        int cx = x1+w;
                        int cy = y1+h;
                        int r = sqrt((cx-x1)*(cx-x1)+(cy-y1)*(cy-y1));
                        if((chk_cir2(cx,cy,r) == true)&&(r > 1))
                        {
                        cam1.pr3[i] = 1;
                        cam1.pr4[j] = 1;

                        printf("Circle:%d %d %d \n",cx,cy,r);
                        getch();
                        }
        }
}
for(i = 0;i <= cam1.NUMBER3; i ++)
for(j = 0;j <= cam1.NUMBER2; j ++)
if((cam1.pr3[i] == 0)&&(cam1.pr2[j] == 0))
{
BOOL BL = false;
for(int k = 1;(k < 2&&BL == false); k ++)
{
        if((cam1.pq3[i][k][0]==cam1.pq2[j][k][0])&&(cam1.pq3[i][k][1] ==-
cam1.pq2[j][k][1]))
                if(cam1.pq3[i][k][1] != 1)
                {
                        BL = false;
                }
                else
                {
                        BL = true;
```

```
        }
}
        if(BL == false)
              // Circle found out ..
        {
              int x1 = cam1.pq3[i][0][0];
              int y1 = cam1.pq3[i][0][1];
              int x2 = x1+cam1.pq3[i][1][0];
              int y2 = y1+cam1.pq3[i][1][1];
              int x3 = cam1.pq2[j][0][0];
              int y3 = cam1.pq2[j][0][1];
              int x4 = x3+cam1.pq2[j][1][0];
              int y4 = y3+cam1.pq2[j][1][1];

              int w = (x2 - x1)/2;
              int h = (y3 - y1)/2;
              int cx = x1+w;
              int cy = y1+h;
              int r = sqrt((cx-x1)*(cx-x1)+(cy-y1)*(cy-y1));
              if((chk_cir2(cx,cy,r) == true)&&(r > 1))
              {
              cam1.pr3[i] = 1;
              cam1.pr2[j] = 1;

                    printf("Circle:%d %d %d \n",cx,cy,r);
                    getch();
              }
        }
}
for(i = 0;i <= cam1.NUMBER1; i ++)
        for(j = 0;j <= cam1.NUMBER4; j ++)
        if((cam1.pr1[i] == 0)&&(cam1.pr4[j] == 0))
        {
        BOOL BL = false;
        for(int k = 1;(k < 2&&BL == false); k ++)
        {

        if((cam1.pq1[i][k][0]==cam1.pq4[j][k][0])&&(cam1.pq1[i][k][1] ==-
cam1.pq4[j][k][1]))
              if(cam1.pq1[i][k][1] != 1)
              {
                    BL = false;
              }
              else
              {
                    BL = true;
              }
        }
if(BL == false)
        // Circle found out ..
{
        int x1 = cam1.pq1[i][0][0];
        int y1 = cam1.pq1[i][0][1];
        int x2 = x1+cam1.pq1[i][1][0];
        int y2 = y1+cam1.pq1[i][1][1];
        int x3 = cam1.pq4[j][0][0];
        int y3 = cam1.pq4[j][0][1];
```

```
int x4 = x3+cam1.pq4[j][1][0];
int y4 = y3+cam1.pq4[j][1][1];

int w = (x2 - x1)/2;
int h = (y3 - y1)/2;
int cx = x1+w;
int cy = y1+h;
int r = sqrt((cx-x1)*(cx-x1)+(cy-y1)*(cy-y1));
if((chk_cir2(cx,cy,r) == true)&&(r > 1))
{
cam1.pr1[i] = 1;
                    cam1.pr4[j] = 1;

                          printf("Circle:%d %d %d \n",cx,cy,r);
                          getch();
                    }
              }
        }


}

BOOL Image::chk_cir1(int x0,int y0,int rad)
{
        int i,j;
        BOOL l = true;

        Circle c(x0,y0,rad);
        for(i = 0; i < 45; i ++)
{
        int x = rad*cos(i*3.14/180);
        int y = c.gety(x);
        int x1 = x0+x;
        int y1 = y0+y;
        if(array[x1][y1] != 1)
              l = false;
        x1 = x0 -x ;
        y1 = y0 +y;
        if(array[x1][y1] != 1)
              l = false;

        x1 = x0 +x ;
        y1 = y0 -y;
              if(array[x1][y1] != 1)
              l = false;

        x1 = x0 -x ;
        y1 = y0 -y;
              if(array[x1][y1] != 1)
              l = false;


        x1 = y0 +y ;
        y1 = x0 +x;
              if(array[x1][y1] != 1)
```

```
                l = false;


        x1 = y0 -y ;
        y1 = x0 +x;
                if(array[x1][y1] != 1)
                l = false;


        x1 = y0 +y ;
                y1 = x0 -x;
                        if(array[x1][y1] != 1)
                        l = false;


                x1 = y0 -y ;
                y1 = x0 -x;
                        if(array[x1][y1] != 1)
                        l = false;


        }

        return 1;
}
BOOL Image::chk_cir2(int x0,int y0,int rad)
{

        return chk_cir1(x0,y0,rad);
}

BOOL Image::find_der(int p,int q)
{
        int i,j;
        float d11,d12,d21,d22,d15;
        float d13,d14,d23,d24;
                d11 = 1.0*cam1.pq1[p][1][1]/cam1.pq1[p][1][0];
                d12 = 1.0*cam1.pq1[p][2][1]/cam1.pq1[p][2][0];
                d21 = abs((d11 + d12)/2.0);
                d22 = (d11 - d12)/(cam1.pq1[p][2][0]);
                float rad1 = abs(pow((1 + pow(d21,2)),1.5)/d22);
                d13 = 1.0*cam1.pq2[q][1][1]/cam1.pq2[q][1][0];
                d14 = 1.0*cam1.pq2[q][2][1]/cam1.pq2[q][2][0];
                d23 = abs((d13 + d14)/2.0);
                d24 = (d13 - d14)/(cam1.pq1[p][2][0]);
                float rad2 = abs(pow((1 + pow(d23,2)),1.5)/d24);

                if(rad1 == rad2)
                {
                        return true;
                }
                else
                {
                        return false;
                }
        }
```

```
int Image::next_brkpt(int in)
{
	int x,y,z;
	cam.put_cam(in,x,y,z);
	char c = z;
	int k;
	switch(c)
	{
		case('A'):
			k = cam.get_2brkpt(in);
			break;
	case('B'):
			k = cam.get_3brkpt(in);
			break;
	case('C'):
			k = cam.get_7brkpt(in);
			break;
	case('D'):
			k = cam.get_0brkpt(in);
			break;
	case('E'):
			k = cam.get_1brkpt(in);
			break;
	case('F'):
			k = cam.get_2brkpt(in);
			break;
	case('G'):
			k = cam.get_0brkpt(in);
			break;
		case('H'):
			k = cam.get_1brkpt(in);
			break;
		default:
			k = -1;
	}
	return k;
}
void Image::find_break(Image1* brk)
// Finds break pts in image and puts in brk
{
	int pt[NO][9];
	pt[0][0] = 0;
	pt[0][1] = 0;
	pt[0][2] = 1;
	pt[0][3] = 1;
	pt[0][4] = 1;
	pt[0][5] = 0;
	pt[0][6] = 0;
	pt[0][7] = 0;
	pt[0][8] = 0;

	pt[1][0] = 0;
	pt[1][1] = 1;
	pt[1][2] = 0;
	pt[1][3] = 0;
	pt[1][4] = 1;
	pt[1][5] = 0;
```

```
pt[1][6] = 1;
pt[1][7] = 0;
pt[1][8] = 0;

pt[2][0] = 1;
pt[2][1] = 0;
pt[2][2] = 0;
pt[2][3] = 0;
pt[2][4] = 1;
pt[2][5] = 0;
pt[2][6] = 0;
pt[2][7] = 1;
pt[2][8] = 0;

pt[3][0] = 0;
pt[3][1] = 0;
pt[3][2] = 0;
pt[3][3] = 1;
pt[3][4] = 1;
pt[3][5] = 0;
pt[3][6] = 0;
pt[3][7] = 0;
pt[3][8] = 1;

pt[4][0] = 0;
pt[4][1] = 0;
pt[4][2] = 0;
pt[4][3] = 0;
pt[4][4] = 1;
pt[4][5] = 1;
pt[4][6] = 1;
pt[4][7] = 0;
pt[4][8] = 0;

pt[5][0] = 0;
pt[5][1] = 0;
pt[5][2] = 1;
pt[5][3] = 0;
pt[5][4] = 1;
pt[5][5] = 0;
pt[5][6] = 0;
pt[5][7] = 1;
pt[5][8] = 0;

pt[6][0] = 0;
pt[6][1] = 1;
pt[6][2] = 0;
pt[6][3] = 0;
pt[6][4] = 1;
pt[6][5] = 0;
pt[6][6] = 0;
pt[6][7] = 0;
pt[6][8] = 1;

pt[7][0] = 1;
pt[7][1] = 0;
pt[7][2] = 0;
```

```
pt[7][3] = 0;
pt[7][4] = 1;
pt[7][5] = 1;
pt[7][6] = 0;
pt[7][7] = 0;
pt[7][8] = 0;

pt[8][0] = 0;
pt[8][1] = 0;
pt[8][2] = 1;
pt[8][3] = 0;
pt[8][4] = 1;
pt[8][5] = 0;
pt[8][6] = 1;
pt[8][7] = 0;
pt[8][8] = 0;

pt[9][0] = 1;
pt[9][1] = 0;
pt[9][2] = 0;
pt[9][3] = 0;
pt[9][4] = 1;
pt[9][5] = 0;
pt[9][6] = 0;
pt[9][7] = 0;
pt[9][8] = 1;

pt[10][0] = 0;
pt[10][1] = 0;
pt[10][2] = 0;
pt[10][3] = 1;
pt[10][4] = 1;
pt[10][5] = 1;
pt[10][6] = 0;
pt[10][7] = 0;
pt[10][8] = 0;

pt[11][0] = 0;
pt[11][1] = 1;
pt[11][2] = 0;
pt[11][3] = 0;
pt[11][4] = 1;
pt[11][5] = 0;
pt[11][6] = 0;
pt[11][7] = 1;
pt[11][8] = 0;

int p[9];

for(int i = 1;i < size;i ++)
for(int j = 1;j < size;j ++)
{
        p[0] = array[i-1][j-1];
        p[1] = array[i-1][j];
        p[2] = array[i-1][j+1];
        p[3] = array[i][j-1];
```

```
p[4] = array[i][j];
p[5] = array[i][j+1];
p[6] = array[i+1][j-1];
p[7] = array[i+1][j];
p[8] = array[i+1][j+1];

int temp1,temp2 = 0;
for(temp1 = 0;temp1 < 9;temp1++)
{
        if(p[temp1] == 1)
                temp2 += 1;
        }

        if(temp2 > 3)
        {
                brk->read_value(i,j,'*');
        }
        if(temp2 == 3)
        for(int k = 0;k < NO; k ++)
        {
                if(equal(pt[k],p))
                {
                        brk->read_value(i,j,k-0+'A');
                        break;
                }
         }
    }
//      brk->draw_image();
//      getch();
}

BOOL equal(int* pt,int* p)
{
        for(int i = 0;i < 9;i ++)
        {
                if(pt[i] == 0)
                {
                if(pt[i] != p[i])
                        return false;
                }
                else
                if(p[i] == 0)
                        return false;
         }

         return true;
 }
```

File pro.cpp

```
#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

#define SIZE 100

#define NO 12
#include "d:\sudha\finproj\image.h"


int* next_break(char* brk,int x,int y);
BOOL chk_dir(char* brk,int size,int x0,int y0,int x,int y);
// Checks 8 directions for circle
main()
{
      int sz;
      printf("Enter size :");
      scanf("%d",&sz);
      Image image(sz);
      image.reset();
      clrscr();
      image.put_circle();
      image.put_circle();
//    image.put_circle();
//    image.draw_image();
      getch();
      image.find_circle();
      printf("Program Over \n");
      getch();

      return 0;
}


BOOL chk_dir(char* brk,int size,int x0,int y0,int x,int y)
{
      BOOL l = false;

      int i = abs(x - x0);
      int j = abs(y - y0);

      int x1 = x0 + i;
      int y1 = y0 + j;
      int x2 = x0 - i;
      int y2 = y0 + j;
      int x3 = x0 + i;
      int y3 = y0 - j;
      int x4 = x0 - i;
      int y4 = y0 - j;
      if((x0 == 10)&&(y0 == 10))
      {
            printf("*****x y are %d %d \n",x,y);
```

```
                printf("four pts are %d,%d %d,%d %d,%d %d,%d
\n",x1,y1 x2,y2,x3,y3,x4,y4);
                getch();
        }

        if(
        (x1 > 0)&&(x1 < size)&&(y1 > 0)&&(y1 < size)&&
        (x2 > 0)&&(x2 < size)&&(y2 > 0)&&(y2 < size)&&
        (x3 > 0)&&(x3 < size)&&(y3 > 0)&&(y3 < size)&&
        (x4 > 0)&&(x4 < size)&&(y4 > 0)&&(y4 < size)
        ){
        if(
        (*(brk+x1*SIZE+y1) != '0') &&(*(brk+y1*SIZE+x1) != '0') &&
        (*(brk+x2*SIZE+y2) != '0') &&(*(brk+y2*SIZE+x2) != '0') &&
        (*(brk+x3*SIZE+y3) != '0') &&(*(brk+y3*SIZE+x3) != '0') &&
        (*(brk+x4*SIZE+y4) != '0') &&(*(brk+y4*SIZE+x4) != '0')
        )
        {
                1 = true;
        }
        }

        return 1;
}
```