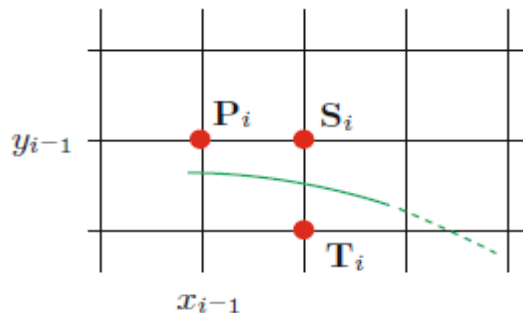


## Bresenham Circle Method

The Bresenham circle method is based on a loop that starts at point  $(0, R)$  and ends at point  $(R/\sqrt{2}, R/\sqrt{2})$  to create one octant of the circle. Each pixel calculated is used to determine seven more pixels, in the remaining seven octants, to create the complete circle. To move from  $(0, R)$  to  $(R/\sqrt{2}, R/\sqrt{2})$ , we need to increment  $x$  and decrement  $y$ . The loop of figure below is set such that the  $x$  coordinate is incremented in every step, but the  $y$  coordinate is only decremented in certain steps (i.e., conditionally). The results are good (i.e., the pixels are fairly uniformly distributed) because in this octant the circle is close to horizontal.



In each step (except the first), the algorithm examines two points, **S** and **T** in figure, that differ only in their  $y$  coordinates, and it selects the one that's closer to the true circle.

For simplicity, we translate the center of circle to  $(0, 0)$ . The algorithm maintains a variable  $d$  (calculated using just additions, subtractions, and shifts) that is updated every step. The sign of  $d$  is used as an indicator, telling the program whether to decrement  $y$  at the step or not. Hence, updating the value of  $d$  in either case is simple and does not require any arithmetic operations beyond addition, subtraction, and shifting. The initial value of  $d$ , is easily found by :

$$d = 3 - 2R.$$

The final algorithm is shown below:

**Algorithm 3: Bresenham circle:** (x, y) circle , R radius

$x=0, y=R, d=3-2 * R$

while  $x < y$  do

    SetPixel ( x , y )

    if  $d > 0$  then

$d = d + 4 * ( x - y ) + 10$

$y = y - 1$

    else

$d = d + 4 * x + 6$

    end if

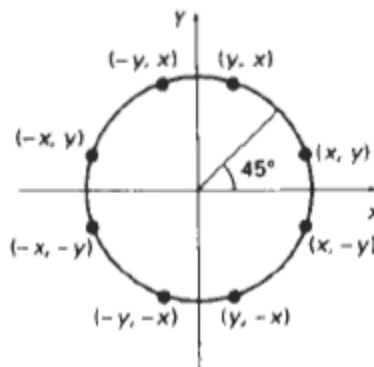
$x = x + 1$

end while

if  $x=y$  then SetPixel( x , y )

**end**

To calculate the value of x , y in the eighth octants, we need to use the symmetry property of circle as in figure:



That will write as function DrawPoint(x, y):

SetPixel (x,y), SetPixel (-x,-y), SetPixel (-x,y), SetPixel (x,-y),

SetPixel (y,x), SetPixel (-y,-x), SetPixel (-y,x), SetPixel (y,-x)

Finally, we must return the circle to the origin center by increase xc to x and decrease yc from y.