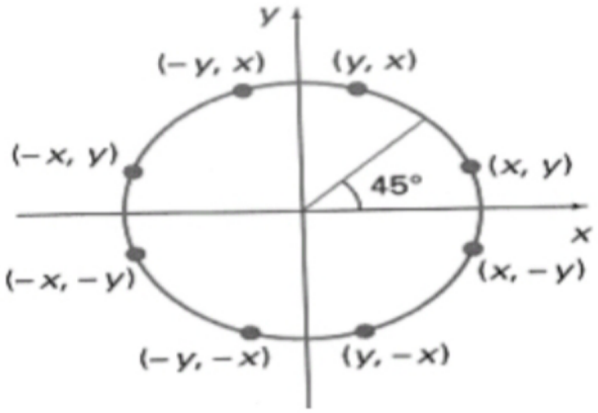
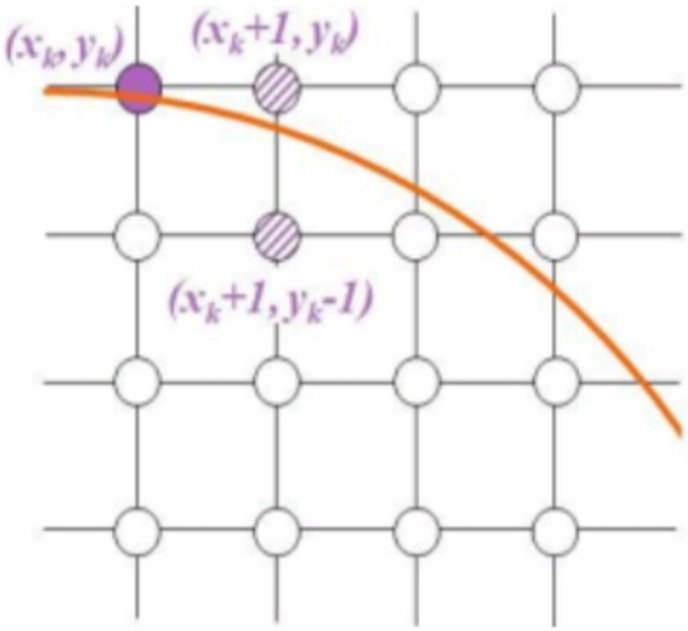
name Pranav Narayan Pisal
roll no 63 div 2 se

Aim: To implement midpoint circle algorithm.
Objective:
Draw a circle using mid-point circle drawing algorithm by determining the points needed for
rasterizing a circle. The mid-point algorithm to calculate all the perimeter points of the circle
in the first octant and then print them along with their mirror points in the other octants.
Theory:
The shape of the circle is similar in each quadrant. We can generate the points in one section
and the points in other sections can be obtained by considering the symmetry about x-axis
and y-axis.



The equation of circle with center at origin is x 2 + y 2 = r 2
Let the circle function is f circle (x, y) - is < 0, if (x, y) is inside circle boundary,



Now the next pixel along the circumference of the circle will be either (xk + 1, yk) or (xk + 1,
yk – 1) whichever is closer the circle boundary.
Let the decision parameter pk is equal to the circle function evaluate at the mid-point between
two pixels.
If pk < 0, the midpoint is inside the circle and the pixel at yk is closer to the circle boundary.
Otherwise, the midpoint is outside or on the circle boundary and the pixel at yk – 1 is closer
to the circle boundary.

Algorithm –

1. Accept the radius ‘r’ and center of the circle (xc
, yc
)
2. Initialize the starting position as (x0
, y0
) = (0, r)
3. Calculate the initial value of decision parameter, p0
as
p0 = 5/4 - r = 1.25 - r

4. At each xk position starting at k = 0
perform the following test
do
{ putpixel (x, y, COLOR)
If pk < 0
{ xnext = xk+1 = xk + 1
ynext = yk+1 = yk
pk+1 = pk + 2xk + 3
}
else
{ xnext = xk+1 = xk + 1
ynext = yk+1 = yk
- 1
pk+1 = pk + 2xk
- 2 yk + 5
}
}
while (x < y)

5. Determine the symmetry points in the other seven octants
.
6. Translate each calculated pixel position in eight octant by
(xc
, yc
)
and plot the pixels.
x = xk+1 + xc
y = yk+1 + yc
putpixel (x, y, COLOUR)
7. Repeat the steps 4 to 6 until x ≥ y.
8. Stop

Program –

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main()
{
int x,y,r,p,xc,yc;
int gd=DETECT,gm;
initgraph(&gd,&gm,"");
printf("enter xc,yc:");
scanf("%d%d",&xc,&yc);
printf("enter r:");
scanf("%d",&r);
x=0;
y=r;
p=1-r;
```

```
do{
putpixel(xc+x,yc+y,RED);
putpixel(xc+y,yc+x,RED);
putpixel(xc+x,yc-x,RED);
putpixel(xc+y,yc-x,RED);
putpixel(xc-x,yc-y,RED);
putpixel(xc-y,yc-x,RED);
putpixel(xc-x,yc+y,RED);
putpixel(xc-y,yc-x,RED);
if(p<0)
{
p=p+(2*x)+3;
}
else
{
y=y-1;

p=p+(2*x)-(2*y)+5;
}
x=x+1;
}
while(y>=x);
getch();
closegraph();
}
```

output –

conclusion

The proposed method for drawing circles is fast and efficient, as it requires only one arc to be drawn and repeated in eight quadrants. This reduces the number of calculations and operations needed to generate a smooth and accurate circle. The method also differs from the traditional line drawing method, which uses multiple short lines to approximate a circle. The line drawing method can result in jagged edges and gaps, especially for small circles or low-resolution displays. Therefore, the proposed method offers a better alternative for drawing circles in computer graphics.