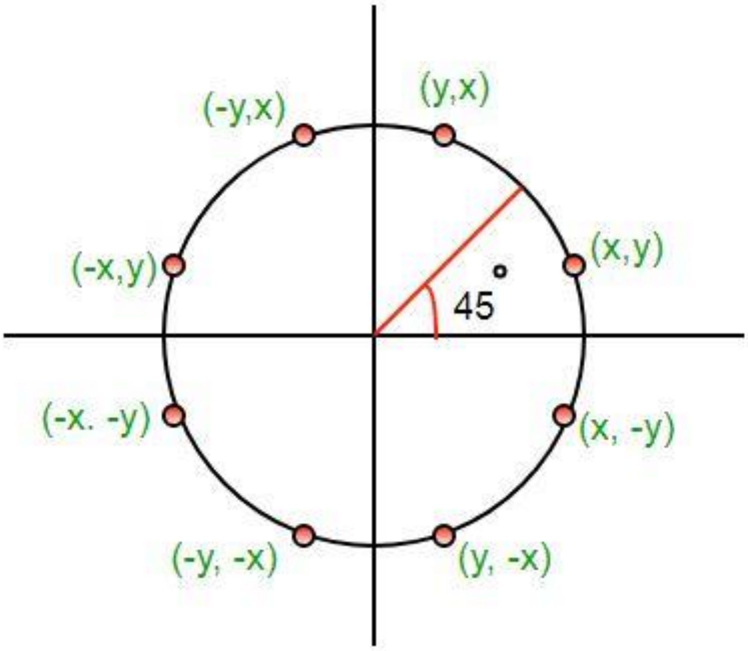
https://www.geeksforgeeks.org/mid-point-circle-drawing-algorithm/

Mid-Point Circle Drawing Algorithm

- Read
- Discuss
- Courses
- Practice
- Video

The **mid-point** circle drawing algorithm is an algorithm used to determine the points needed for rasterizing a circle.

We use 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. This will work because a circle is symmetric about its centre.



The algorithm is very similar to the Mid-Point Line Generation Algorithm. Here, only the boundary condition is different.

For any given pixel (x, y), the next pixel to be plotted is either **(x, y+1)** or **(x-1, y+1)**. This can be decided by following the steps below.

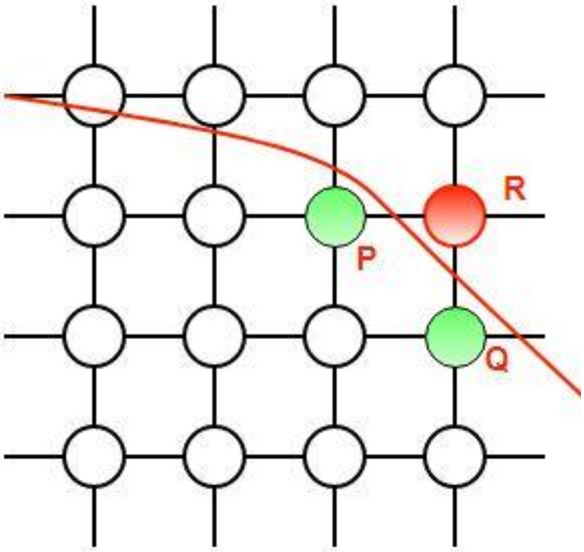
1. Find the mid-point **p** of the two possible pixels i.e (x-0.5, y+1)
2. If **p** lies inside or on the circle perimeter, we plot the pixel (x, y+1), otherwise if it’s outside we plot the pixel (x-1, y+1)

Boundary Condition : Whether the mid-point lies inside or outside the circle can be decided by using the formula:-

Given a circle centered at (0,0) and radius r and a point p(x,y)

$F(p) = x^2 + y^2 – r^2$

if $F(p)<0$, the point is inside the circle
$F(p)=0$, the point is on the perimeter
$F(p)>0$, the point is outside the circle



In our program, we denote F(p) with P. The value of P is calculated at the mid-point of the two contending pixels i.e. (x-0.5, y+1). Each pixel is described with a subscript k.

$P_k = (X_k — 0.5)^2 + (y_k + 1)^2 – r^2$

Now,

$x_{k+1} = x_k$ or x_{k-1} , $y_{k+1}= y_k +1$

 $P_{k+1} = (x_{k+1} – 0.5)^2 + (y_{k+1} +1)^2 – r^2$

$= (x_{k+1} – 0.5)^2 + [(y_k +1) + 1]^2 – r^2$

$= (x_{k+1} – 0.5)^2 + (y_k +1)^2 + 2(y_k + 1) + 1 – r^2$

$= (x_{k+1} – 0.5)^2 + [– (x_k – 0.5)^2 +(x_k – 0.5)^2] + (y_k + 1)^2 – r^2 + 2(y_k + 1) + 1$

$= P_k + (x_{k+1} – 0.5)^2 – (x_k – 0.5)^2 + 2(y_k + 1) + 1$

$= P_k + (x^2_{k+1} – x^2_k) – (x_{k+1} – x_k) + 2(y_k + 1) + 1$

$= P_k + 2(y_k +1) + 1$, when $P_k <=0$ i.e the midpoint is inside the circle

$(x_{k+1} = x_k)$

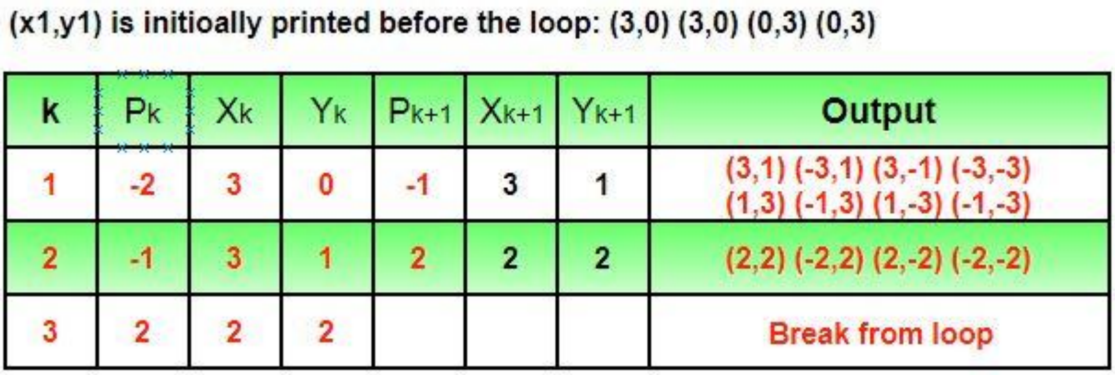
$P_k + 2(y_k +1) – 2(x_k – 1) + 1$, when $P_k>0$ I.e the mid point is outside the circle($x_{k+1} = x_k - 1$)

The first point to be plotted is (r, 0) on the x-axis. The initial value of P is calculated as follows:-

$P1 = (r – 0.5)^2 + (0+1)^2 – r^2$
$= 1.25 – r$
$= 1 -r$ (When rounded off)

Examples:

```
Input : Centre -> (0, 0), Radius -> 3
Output : (3, 0) (3, 0) (0, 3) (0, 3)
         (3, 1) (-3, 1) (3, -1) (-3, -1)
         (1, 3) (-1, 3) (1, -3) (-1, -3)
         (2, 2) (-2, 2) (2, -2) (-2, -2)
```



(x1,y1) is initioally printed before the loop: (3,0) (3,0) (0,3) (0,3)

k	P_k	X_k	Y_k	P_{k+1}	X_{k+1}	Y_{k+1}	Output
1	-2	3	0	-1	3	1	(3,1) (-3,1) (3,-1) (-3,-3) (1,3) (-1,3) (1,-3) (-1,-3)
2	-1	3	1	2	2	2	(2,2) (-2,2) (2,-2) (-2,-2)
3	2	2	2				Break from loop

```
 void KVectorUtil::MidpointCircle(HDC hdc, int x_centre, int y_centre, int r,
Gdiplus::Color color)
{
    int x = r;
    int y = 0;

    Gdiplus::Color color2 = color;
#ifdef _DEBUG
    //color2 = Gdiplus::Color::Red; // for debug
#endif

    // Printing the initial point on the axes
    // after translation
    PutPixel(hdc, x + x_centre, y + y_centre, color);

    // When radius is zero only a single
    // point will be printed
    if (r > 0)
    {
        PutPixel(hdc, -x + x_centre, y + y_centre, color2);
        PutPixel(hdc, y + x_centre, x + y_centre, color);
        PutPixel(hdc, -y + x_centre, -x + y_centre, color2);
    }

    // Initialising the value of P
    int P = 1 - r;
    int dbgCnt = 0;
    while (x > y)
    {
        y++;

        // Mid-point is inside or on the perimeter
        if (P <= 0)
            P = P + 2 * y + 1;
        // Mid-point is outside the perimeter
        else
        {
            x--;
            P = P + 2 * y - 2 * x + 1;
        }

        // All the perimeter points have already been printed
        if (x < y)
            break;

#ifdef _DEBUG
        //if (dbgCnt == g_idebug)
        //    break;
        //dbgCnt += 1;
#endif

        // Printing the generated point and its reflection
        // in the other octants after translation
        PutPixel(hdc, x + x_centre, y + y_centre, color);
        PutPixel(hdc, -x + x_centre, y + y_centre, color2);
        PutPixel(hdc, x + x_centre, -y + y_centre, color);
        PutPixel(hdc, -x + x_centre, -y + y_centre, color2);

        // If the generated point is on the line x = y then
        // the perimeter points have already been printed
        if (x != y)
        {
            PutPixel(hdc, y + x_centre, x + y_centre, color);
            PutPixel(hdc, -y + x_centre, x + y_centre, color2);
            PutPixel(hdc, y + x_centre, -x + y_centre, color);
            PutPixel(hdc, -y + x_centre, -x + y_centre, color2);
        }
    }
}
```