SCAN CONVERTING

LINES and CIRCLES

LINE DRAWING

Description: Given the specification for a straight line, find the collection of addressable pixels which most closely approximates this line.

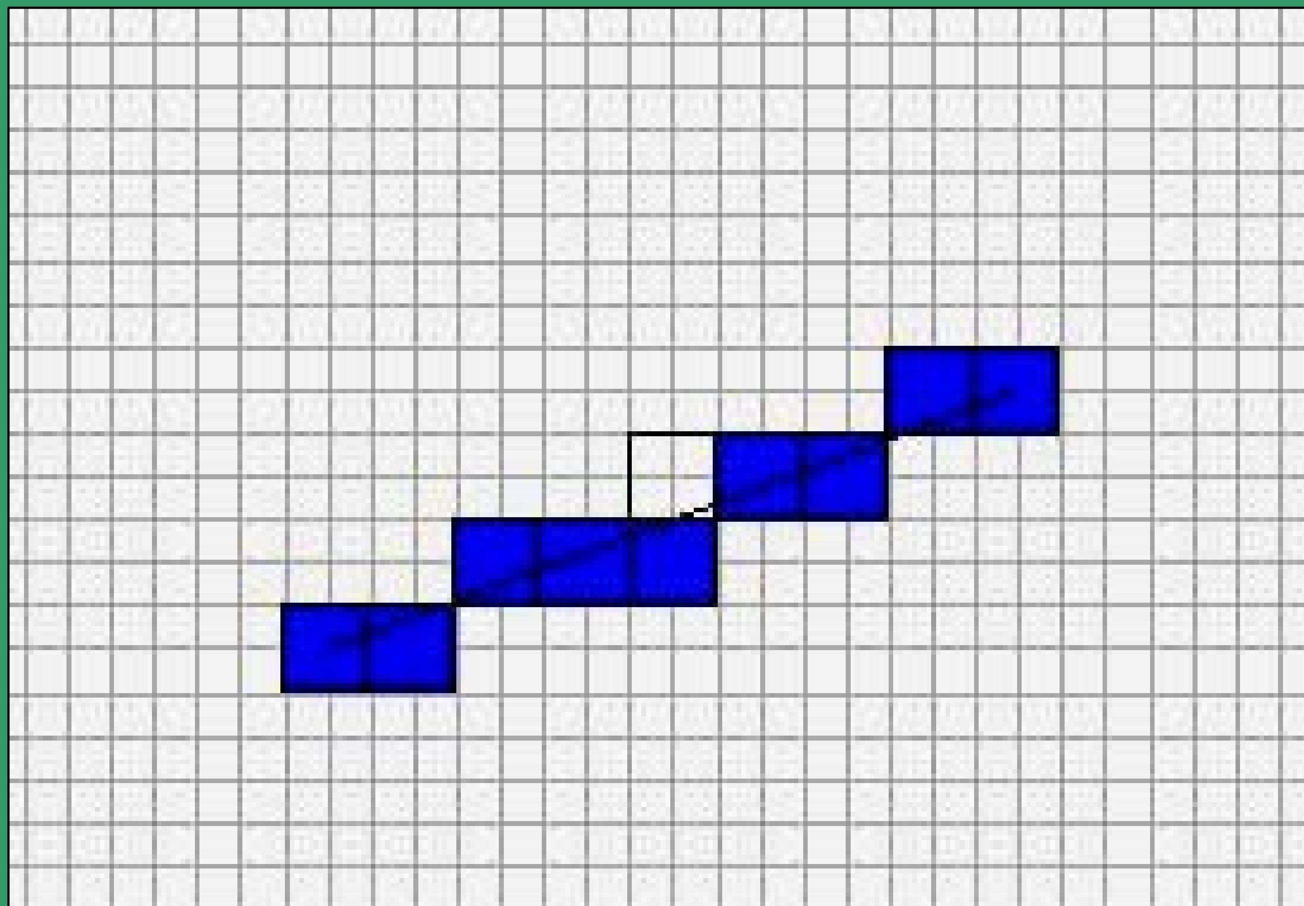
Goals (not all of them are achievable with the discrete space of a raster device):

- Straight lines should appear straight.
- Lines should start and end accurately, matching endpoints with connecting lines.
- Lines should have constant brightness.
- Lines should be drawn as rapidly as possible.

Problems:

- How do we determine which pixels to illuminate to satisfy the above goals?
- Vertical, horizontal, and lines with slope = +/- 1, are easy to draw.
- Others create problems: stair-casing/ jaggies/aliasing.
- Quality of the line drawn depends on the location of the pixels and their brightness

It is difficult to determine whether a pixel belongs to an object



Direct Solution:

Solve $y=mx+b$, where $(0,b)$ is the y-intercept and m is the slope.

Go from x_0 to x_1:

calculate round(y) from the equation.

Take an example, $b = 1$ (starting point $(0,1)$) and $m = 3/5$.

Then $x = 1, y = 2 = \mathit{round}(8/5)$

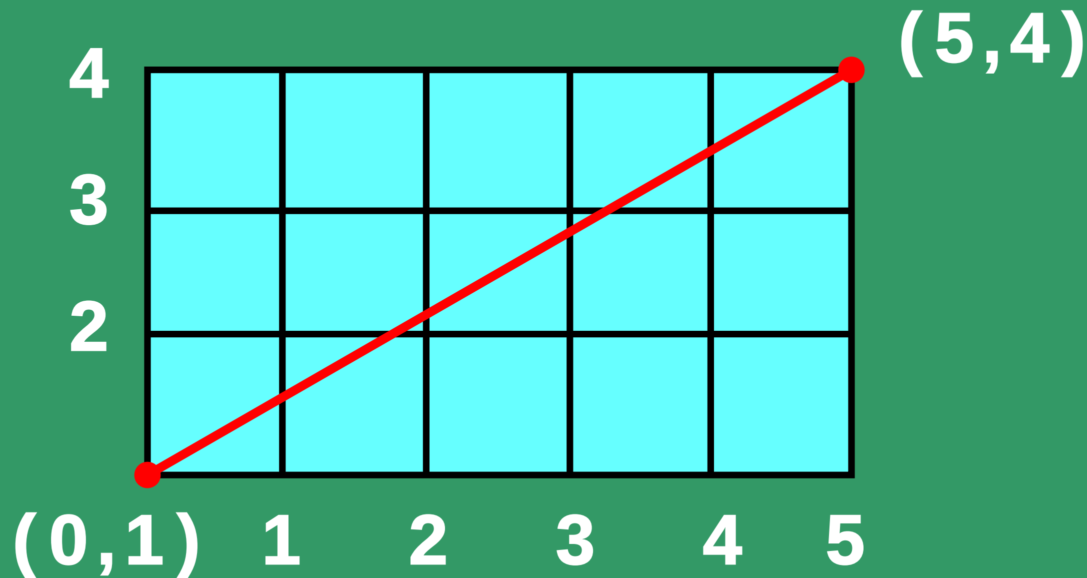
$x = 2, y = 2 = \mathit{round}(11/5)$

$x = 3, y = 3 = \mathit{round}(14/5)$

$x = 4, y = 3 = \mathit{round}(17/5)$

$x = 5, y = 4 = \mathit{round}(20/5)$

For results, see next slide.



Ideal Case of a line drawn in a graph paper

Choice of pixels in the raster, as integer values

x = 1, y = 2 = *round(8/5)*

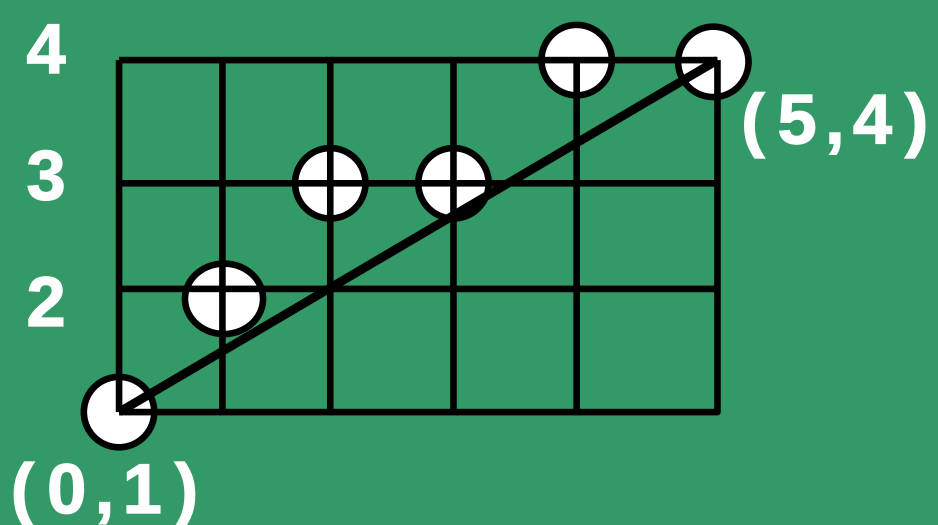
x = 2, y = 2 = *round(11/5)*

x = 3, y = 3 = *round(14/5)*

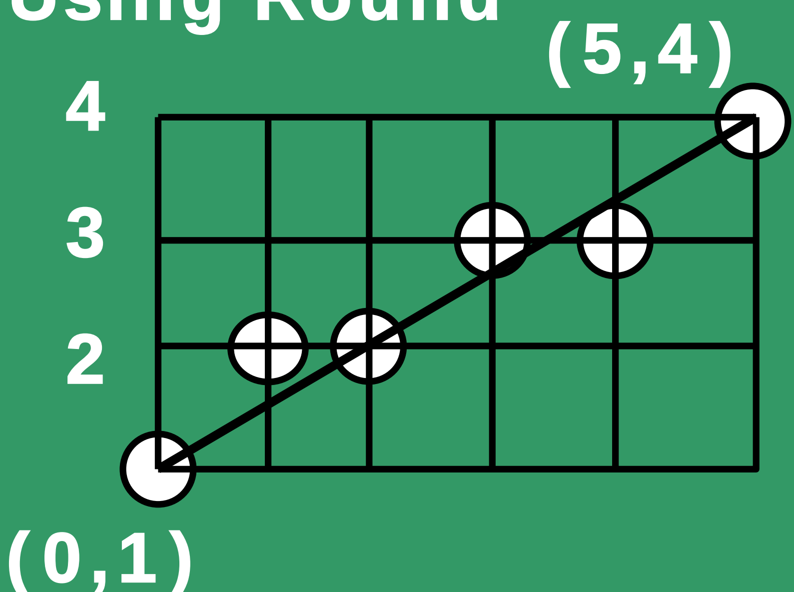
x = 4, y = 3 = *round(17/5)*

x = 5, y = 4 = *round(20/5)*

Using next highest



Using Round



Why is this undesired?

- \`*´ and \`/´ are expensive
- Round() function needed
- Can get gaps in the line (if slope > 1)

Take another example:

$$y = 10.x + 2$$

$$x=1, y=12;$$

$$x=2, y=22.$$

DDA - Digital Difference Analyzer

Incremental Algorithm.

Based on $y = (y_1 - y_0) / (x_1 - x_0)\, x + b$

Assume $x_1 > x_0$ and $|dx| > |dy|$

(can be easily modified for the other cases.)

The Algorithm:

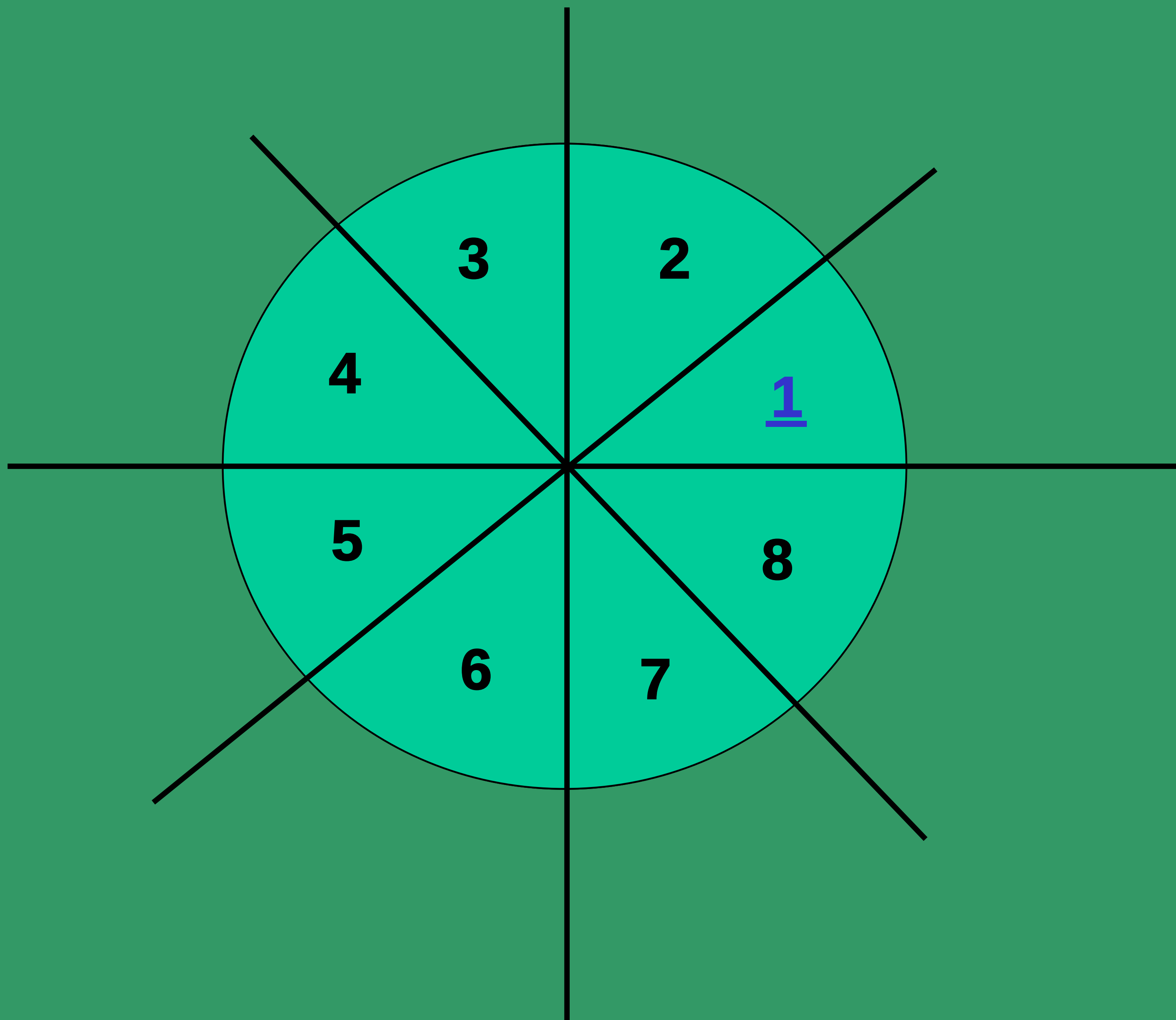
```
dx = x1 - x0 ;
dy = y1 - y0 ;
 m = dy/dx ;
 y = y0 ;
 for (x=x0 to x1)
     draw_point (x, round(y)) ;
     y=y+m;
 end for
```

Problems:

Still uses floating point and round() inside the loop.

How can we get rid of these?

Octants covering the 2-D space



MIDPOINT LINE ALGORITHM

Incremental Algorithm (Assume first octant)

Given the choice of the current pixel, which one do we choose next : E or NE?

Equations:

1. $y = (dy/dx) * x + B$

Rewrite as:

2. $F(x,y) = a*x + b*y + c = 0$

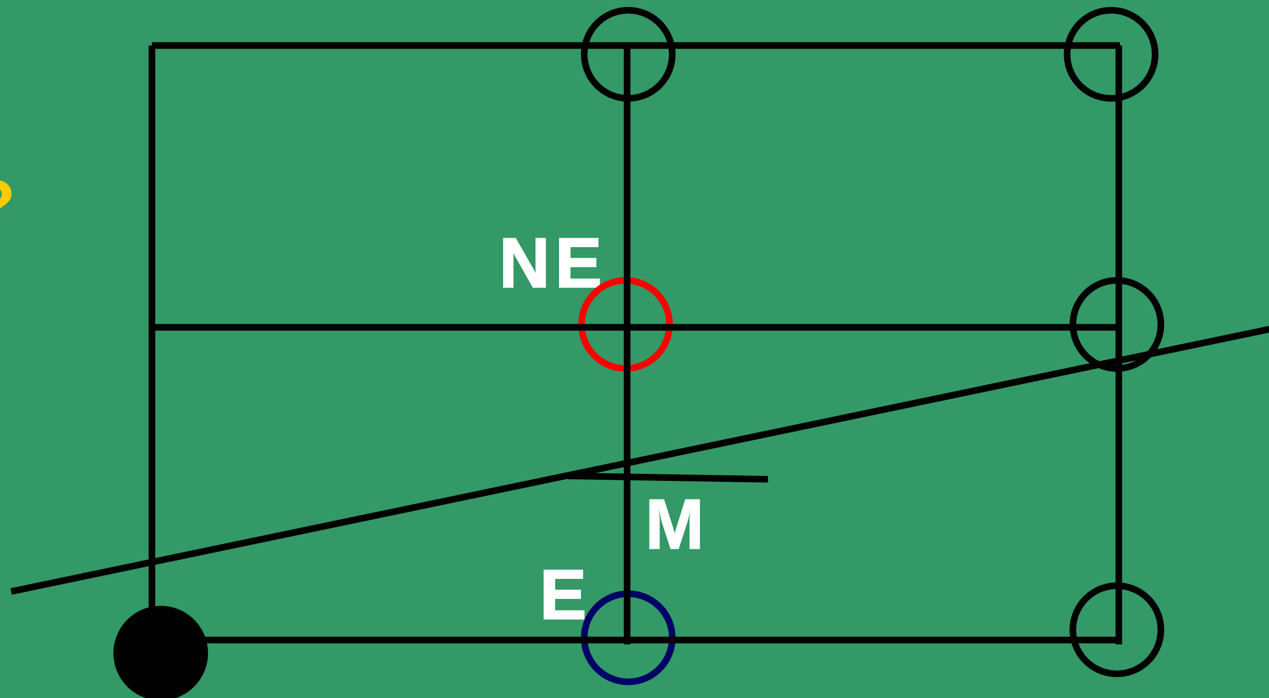
Gives: $F(x,y) = dy*x - dx*y + B*dx = 0$

$\Rightarrow a = dy, b = -dx, c = B*dx$

Criteria:

Evaluate the mid-point, M, w.r.t. the equation of the line.

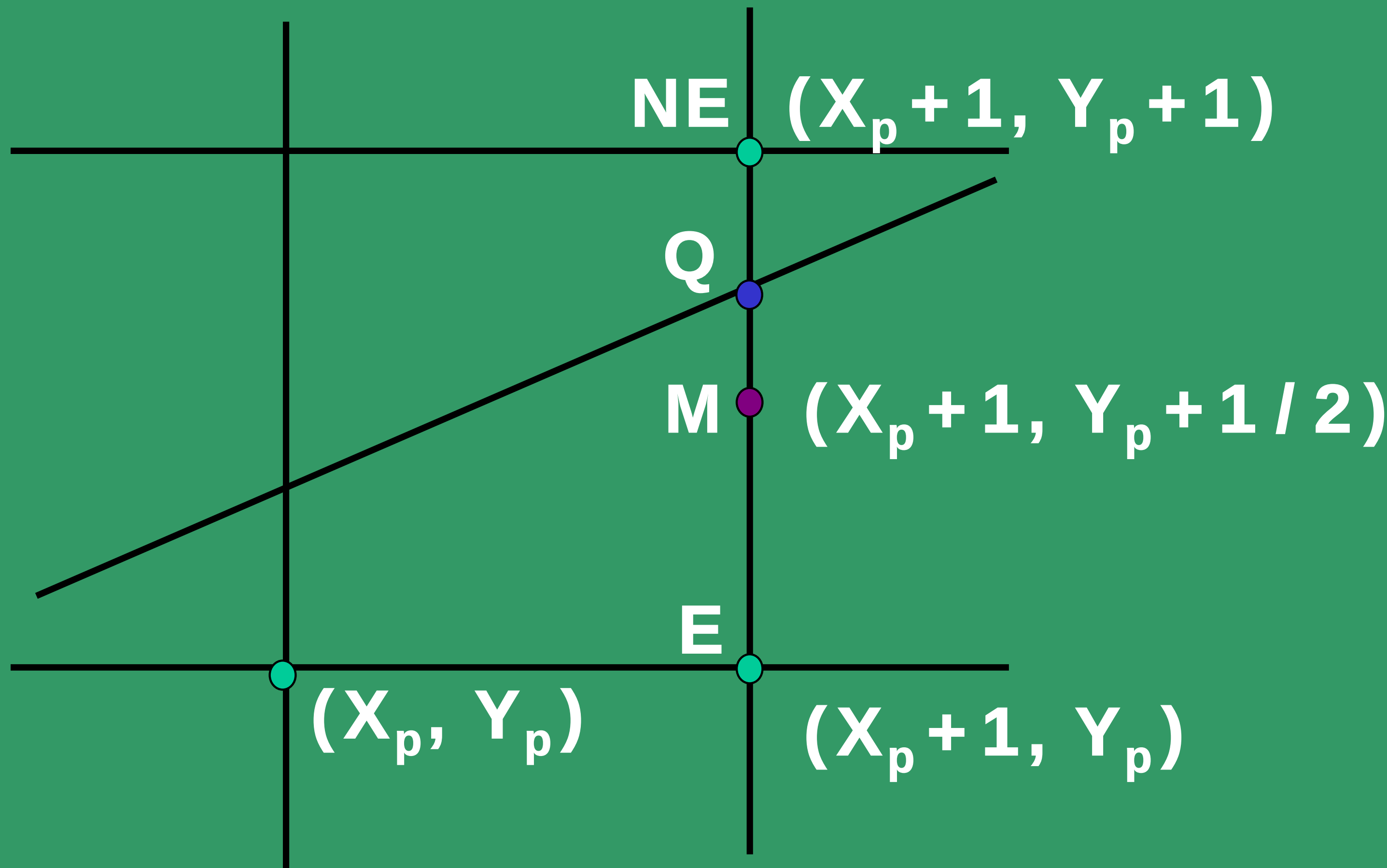
Choice: E or NE?



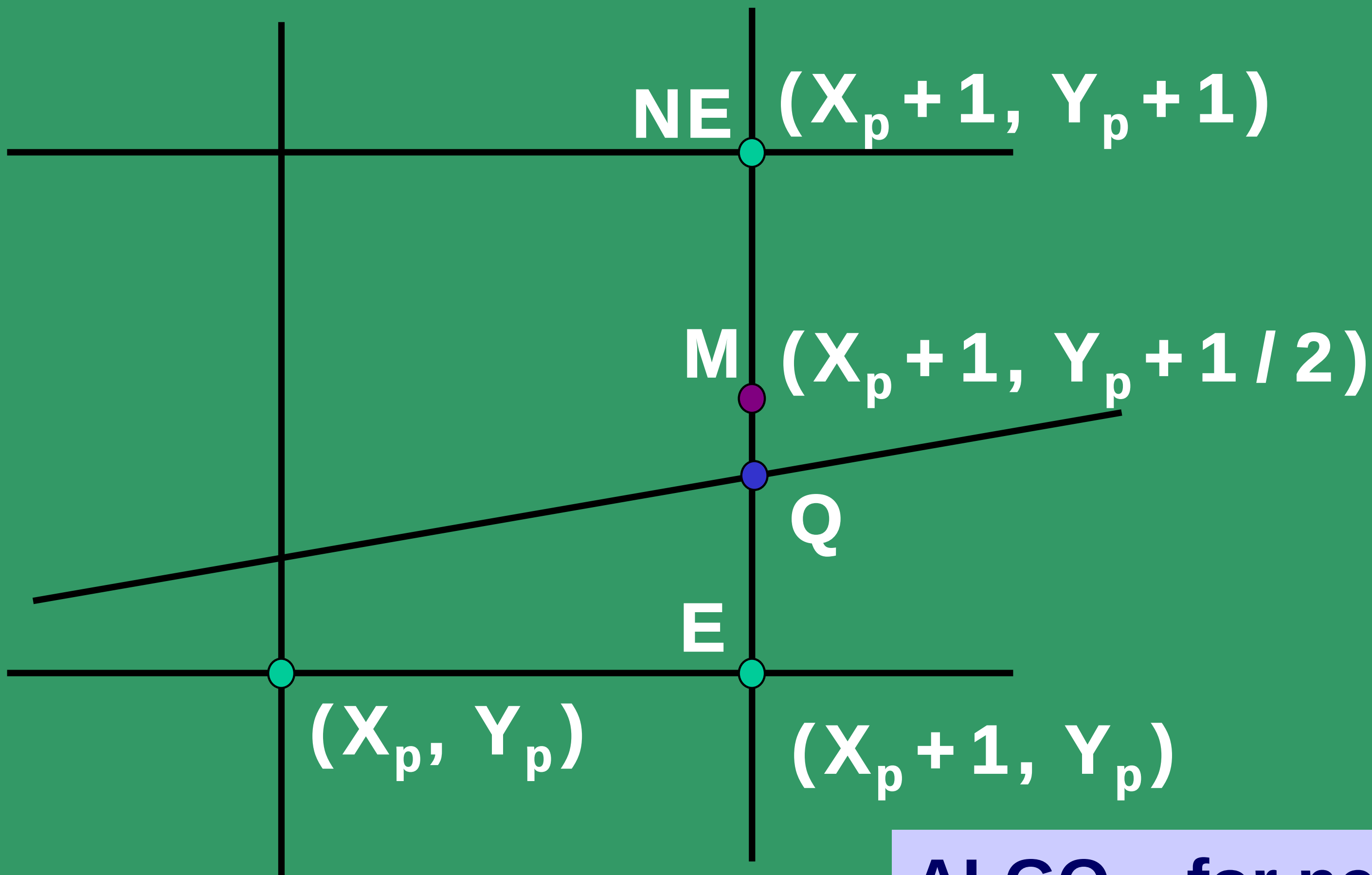
F(x,y) = dy*x - dx*y + B*dx = 0

F(x,y) > 0; if point below the line

F(x,y) < 0; if point above the line



Q is above M,
hence select NE pixel as your next choice



Q is below M, hence select E pixel as your next choice

ALGO – for next choice:

If F(M) > 0 ***/*Q is above M */***

then Select NE

/*M is below the line*/

else Select E ;

/* also with F(M) = 0 */

Evaluate mid-point M using a decision variable $d = F(X,Y)$;

$d = F(X_p+1, Y_p+1/2) = a(X_p+1)+b(Y_p+1/2)+c;$ at M,

Set $d_{old} = d$;

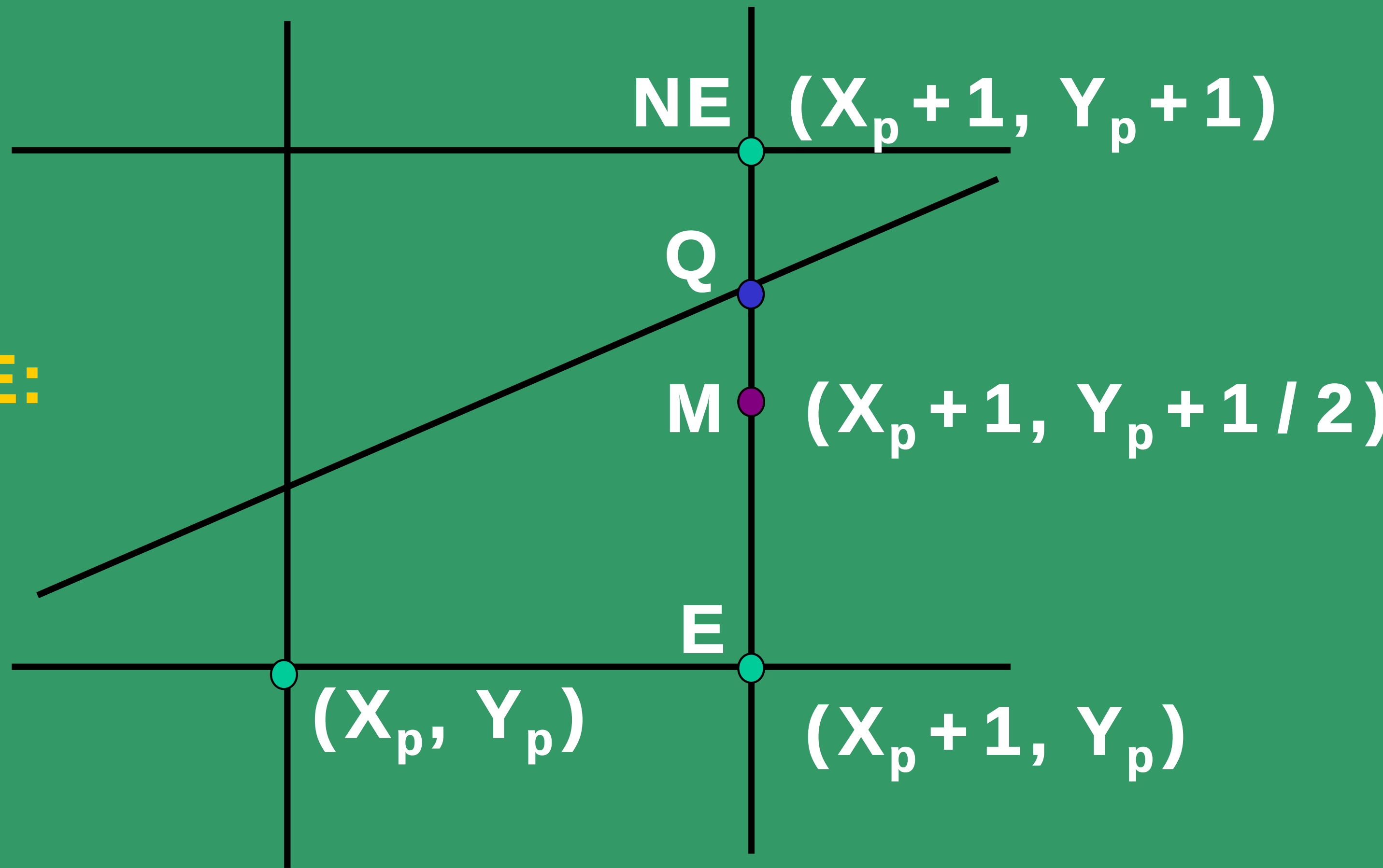
Based on the sign of d, you choose E or NE.

Case I. Chosen E:

$$d_{new} = F(X_p+2, Y_p+1/2)$$
$$= a(X_p+2) + b(Y_p+1/2) + c$$

$$(\Delta d)_E = d_{new} - d_{old} = a \quad /* = dy */$$

Case II.
Chosen NE:



$$d_{new} = F(X_p+2, Y_p+3/2)$$
$$= a(X_p+2) + b(Y_p+3/2) + c$$

$(\Delta d)_{NE} = d_{new} - d_{old} = a + b$ /* $= dy - dx$ */

Update using $d_{new} = d_{old} + \Delta d$

Midpoint criteria

$d = F(M) = F(X_p + 1, Y_p + 1/2);$

if $d > 0$ choose NE

else */* if d <= 0 */* choose E ;

Case EAST :

increment M by 1 in x

$d_{new} = F(M_{new}) = F(X_p + 2, Y + 1/2)$

$(\Delta d)_E = d_{new} - d_{old} = a = dy$

$(\Delta d)_E = dy$

Case NORTH-EAST:

increment M by 1 in both x and y

$d_{new} = F(M_{new}) = F(X_p + 2, Y_p + 3/2)$

$(\Delta d)_{NE} = d_{new} - d_{old} = a + b = dy - dx$

$(\Delta d)_{NE} = dy - dx$

What is d_{start}?

$$d_{start} = F(x_0 + 1, y_0 + 1/2)$$
$$= ax_0 + a + by_0 + b/2 + c$$
$$= F(x_0, y_0) + a + b/2$$
$$= dy - dx/2$$

Let's get rid of the fraction and see what we end up with for all the variables:

$$d_{start} = 2dy - dx;$$

$$(\Delta d)_E = 2dy;$$

$$(\Delta d)_{NE} = 2(dy - dx);$$

The Midpoint Line Algorithm

$x = x_0;$ $y = y_0;$

$dy = y_1 - y_0;$ $dx = x_1 - x_0;$

$d = 2dy - dx;$

$(\Delta d)_E = 2dy;$

$(\Delta d)_{NE} = 2(dy - dx);$

```
Plot_Point(x,y)
```

The Midpoint Line Algorithm (Contd.)

```
while (x < x1)
        if (d <= 0)     /* Choose E */
              d = d + (Δd)E ;

        else            /* Choose NE */
              d = d + (Δd)NE ;
              y = y + 1

        endif

        x = x + 1 ;

        Plot_Point(x, y) ;

end while
```

Example:

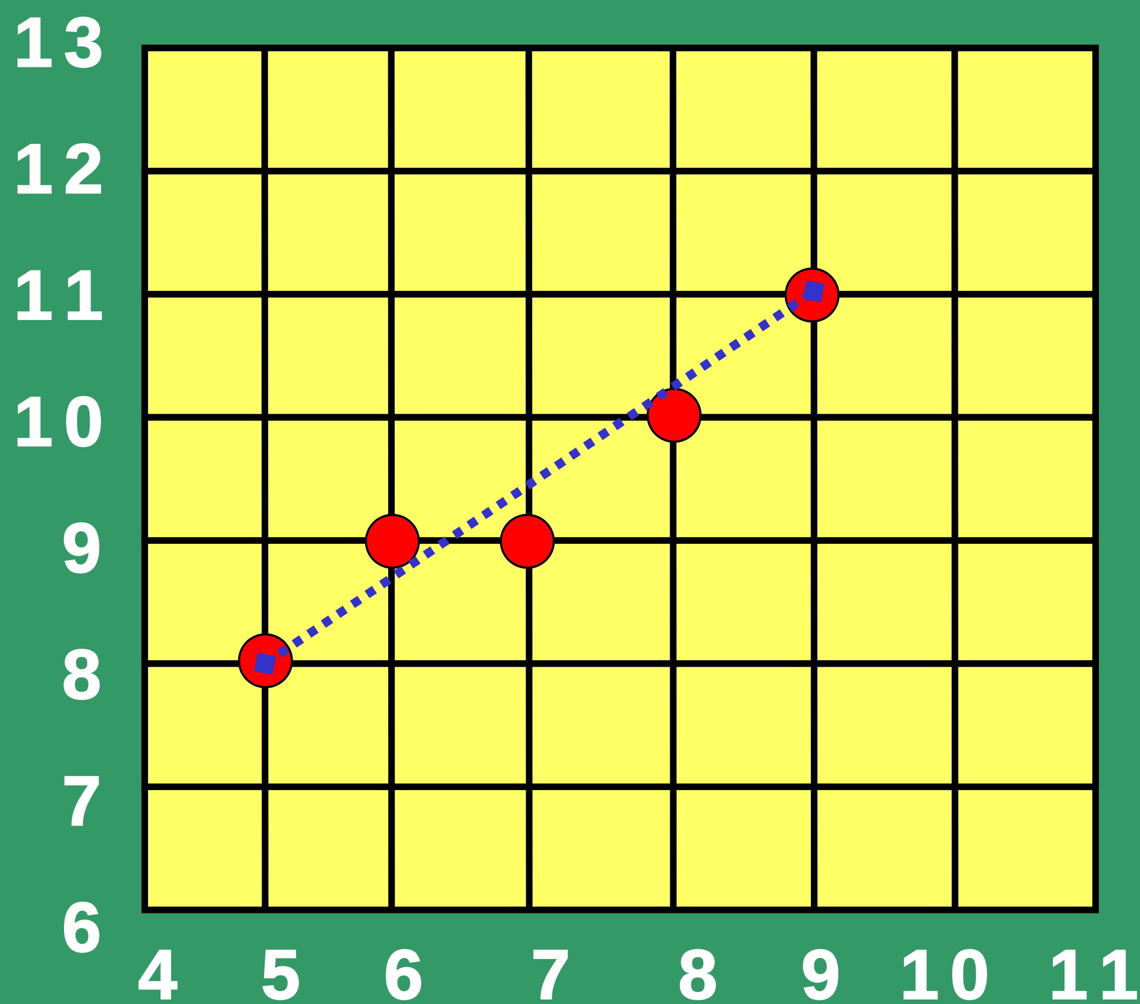
Starting point: (5, 8)
Ending point: (9, 11)

Successive steps:

- d=2, (6, 9)
- d=0, (7, 9)
- d=6, (8, 10)
- d=4, (9, 11)

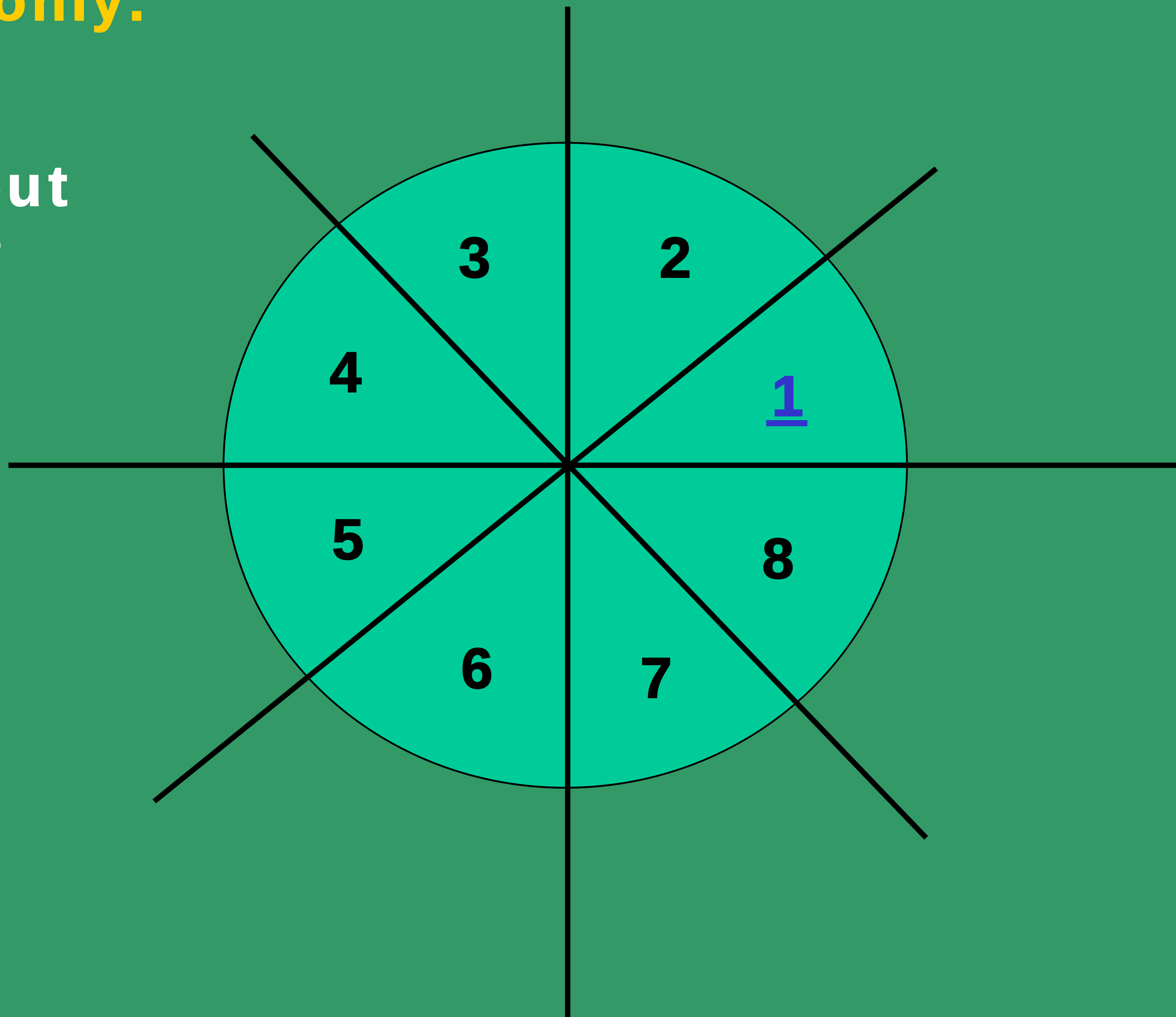
INIT: $dy = 3$; $dx = 4$; $d_{start} = 2$;

$(\Delta d)_E = 6$; $(\Delta d)_{NE} = -2$;



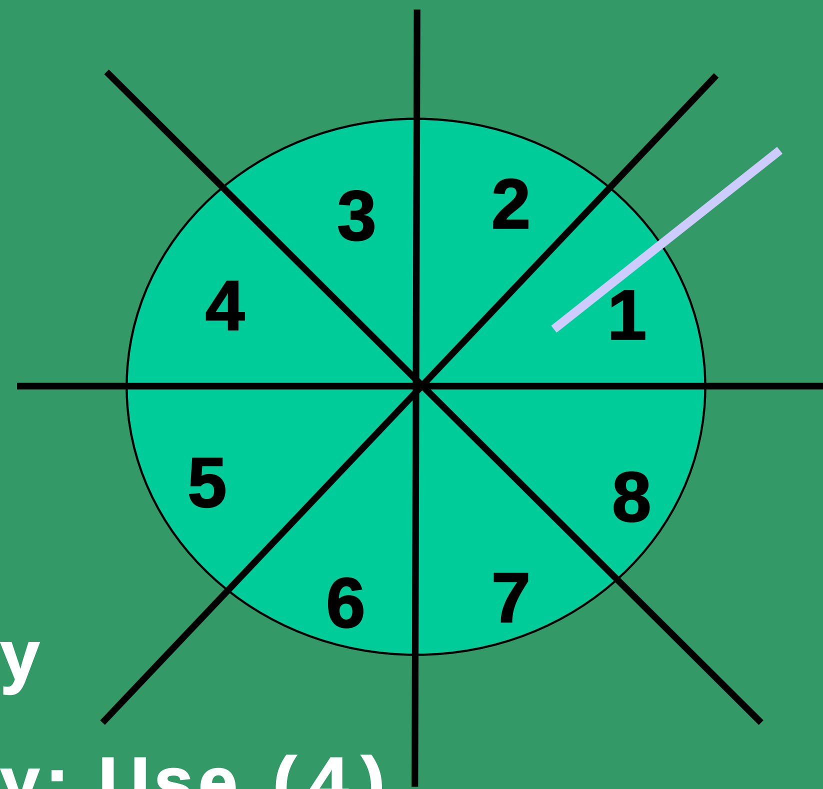
We have considered lines in the first Quadrant only.

What about the rest?



How do you generalize this to the other octants?

Octant	Change
1	none
2	Switch roles of x & y
3	Switch roles of x & y; Use (4)
4	Draw from P_1 to P_0; Use (8)
5	Draw from P_1 to P_0
6	Draw from P_1 to P_0; Use (2)
7	Switch roles of x & y; Use (8)
8	Use y = y - 1.



Octant	Change
1	None
2	Switch roles of x & y
3	Switch roles of x & y; Use (4)
4	Draw from P_1 to P_0; Use (8)
5	Draw from P_1 to P_0
6	Draw from P_1 to P_0; Use (2)
7	Switch roles of x & y; Use (8)
8	Use y = y - 1.

Draw from P_1 to P_0:

swap(P_0, P_1).

Use y = y - 1; dy = -dy;

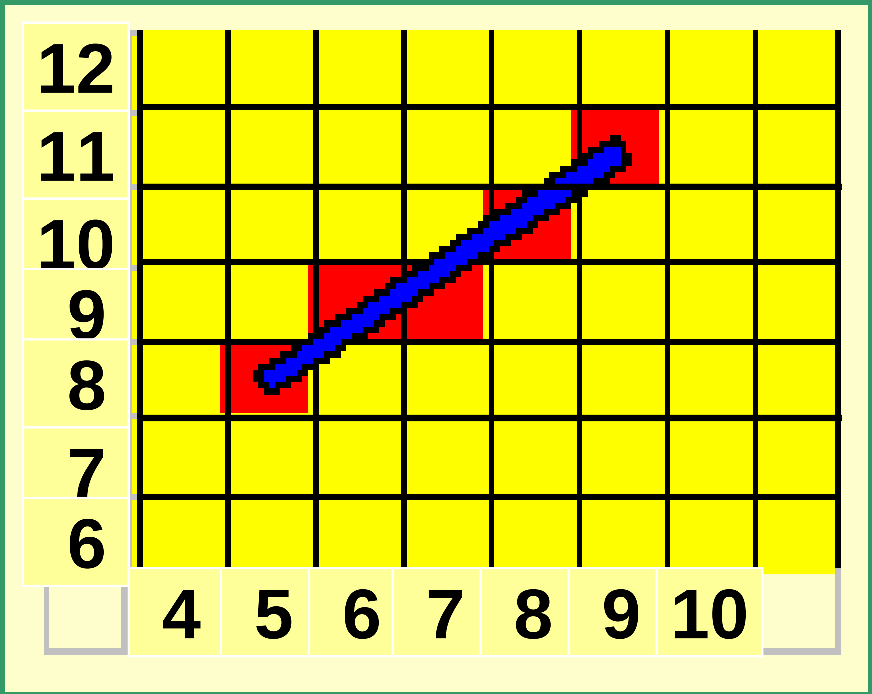
Switch Roles of X & Y:

Swap (x_1, y_1);

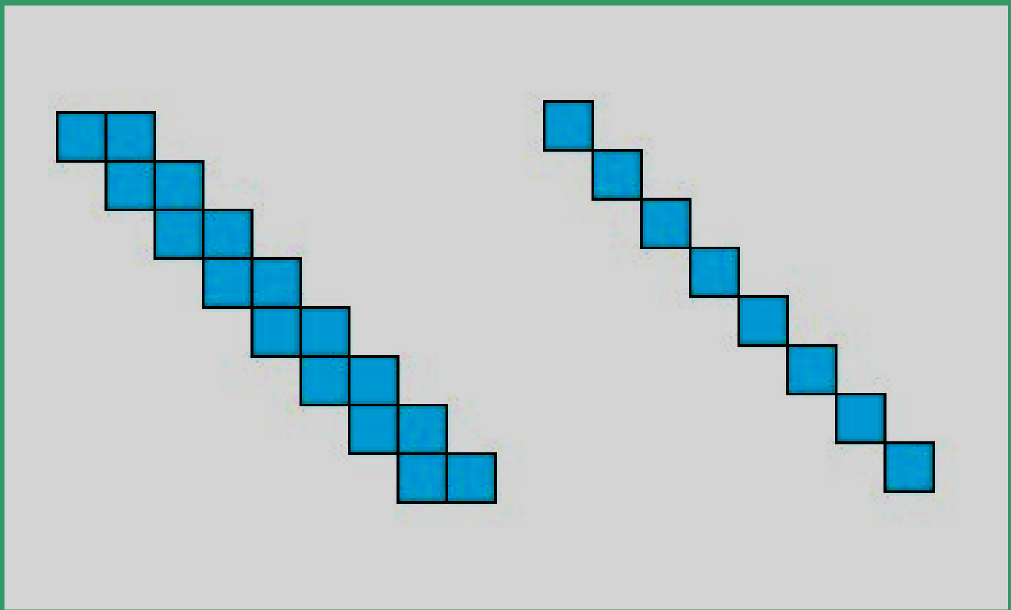
Swap (x_0, y_0);

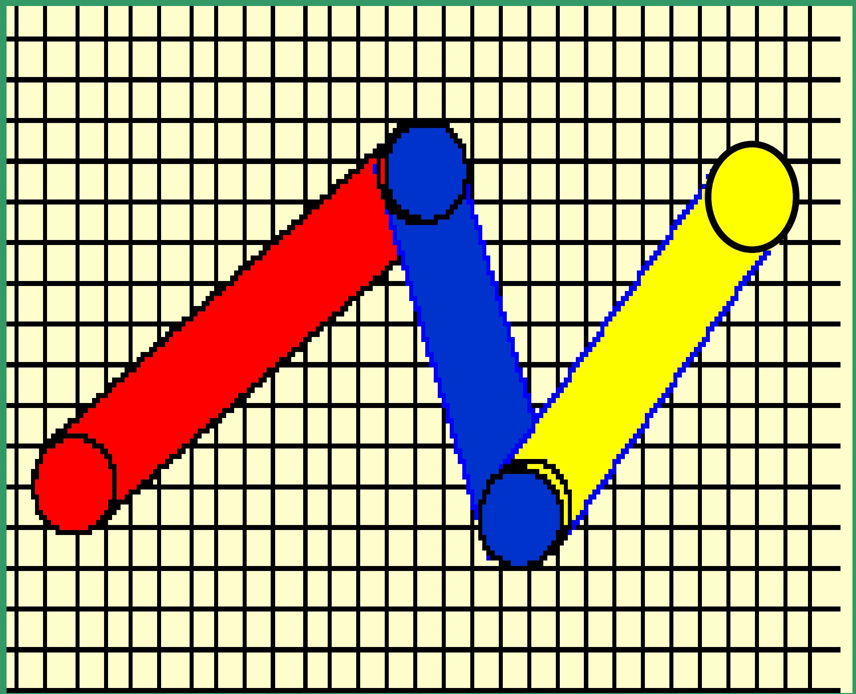
Swap (dx, dy);

plot_point(y, x);

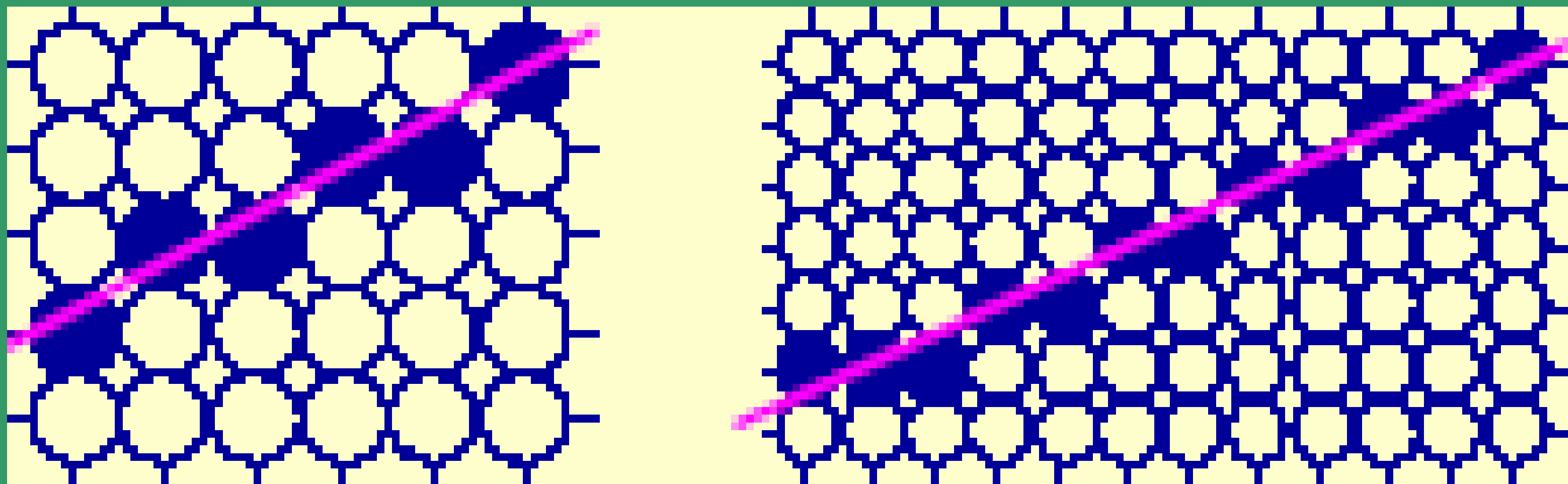


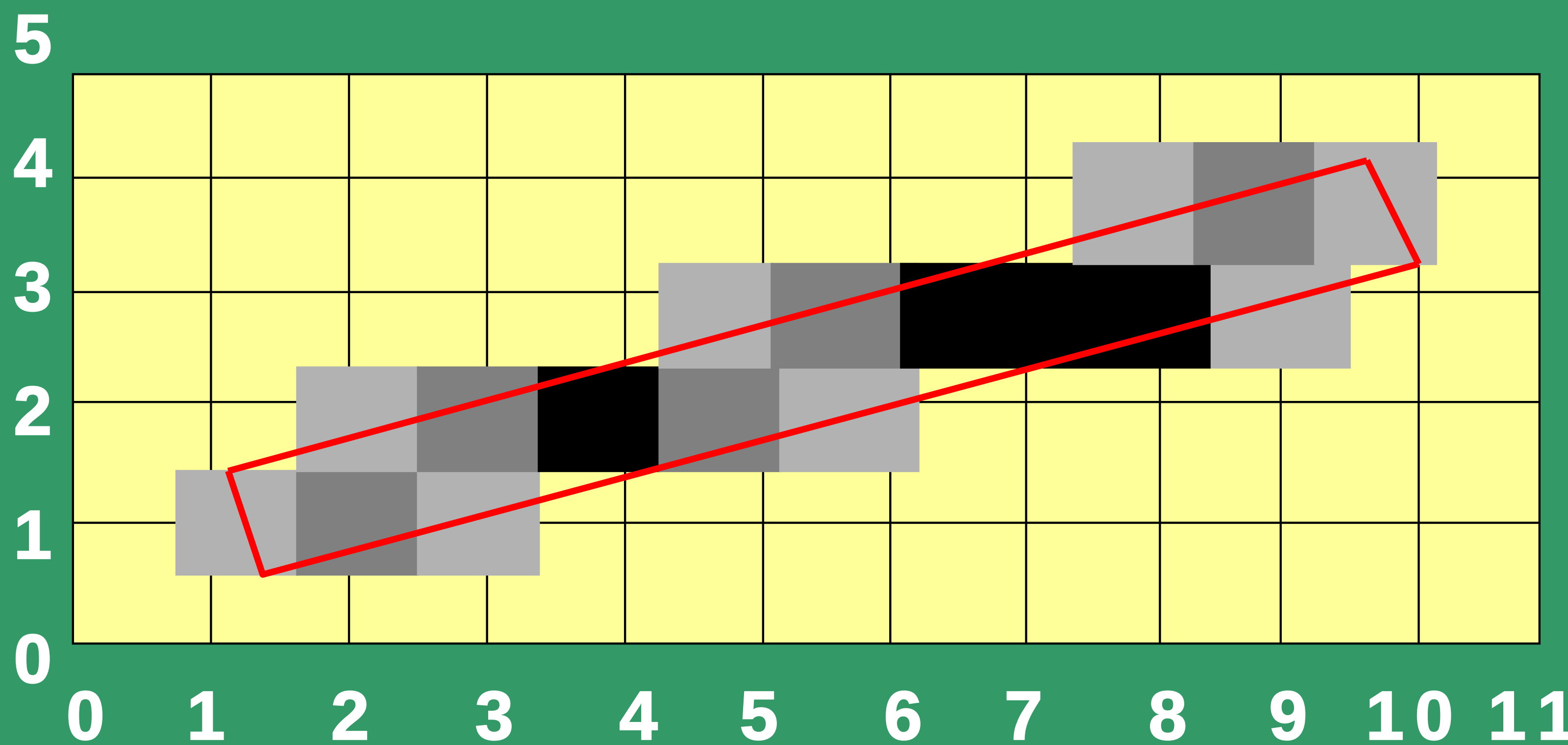
Issues: Staircasing, Fat lines, end-effects and end-point ordering.

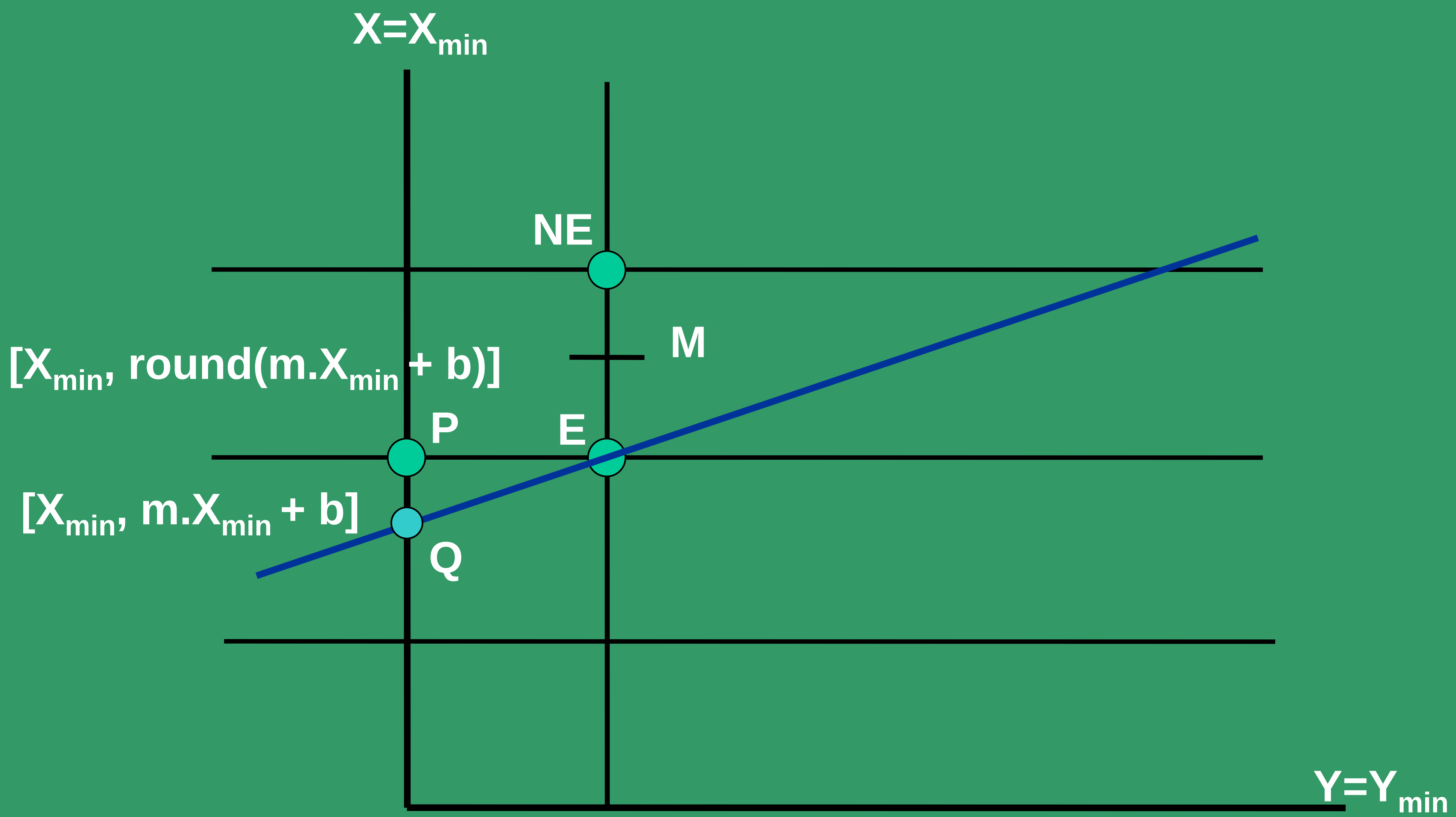




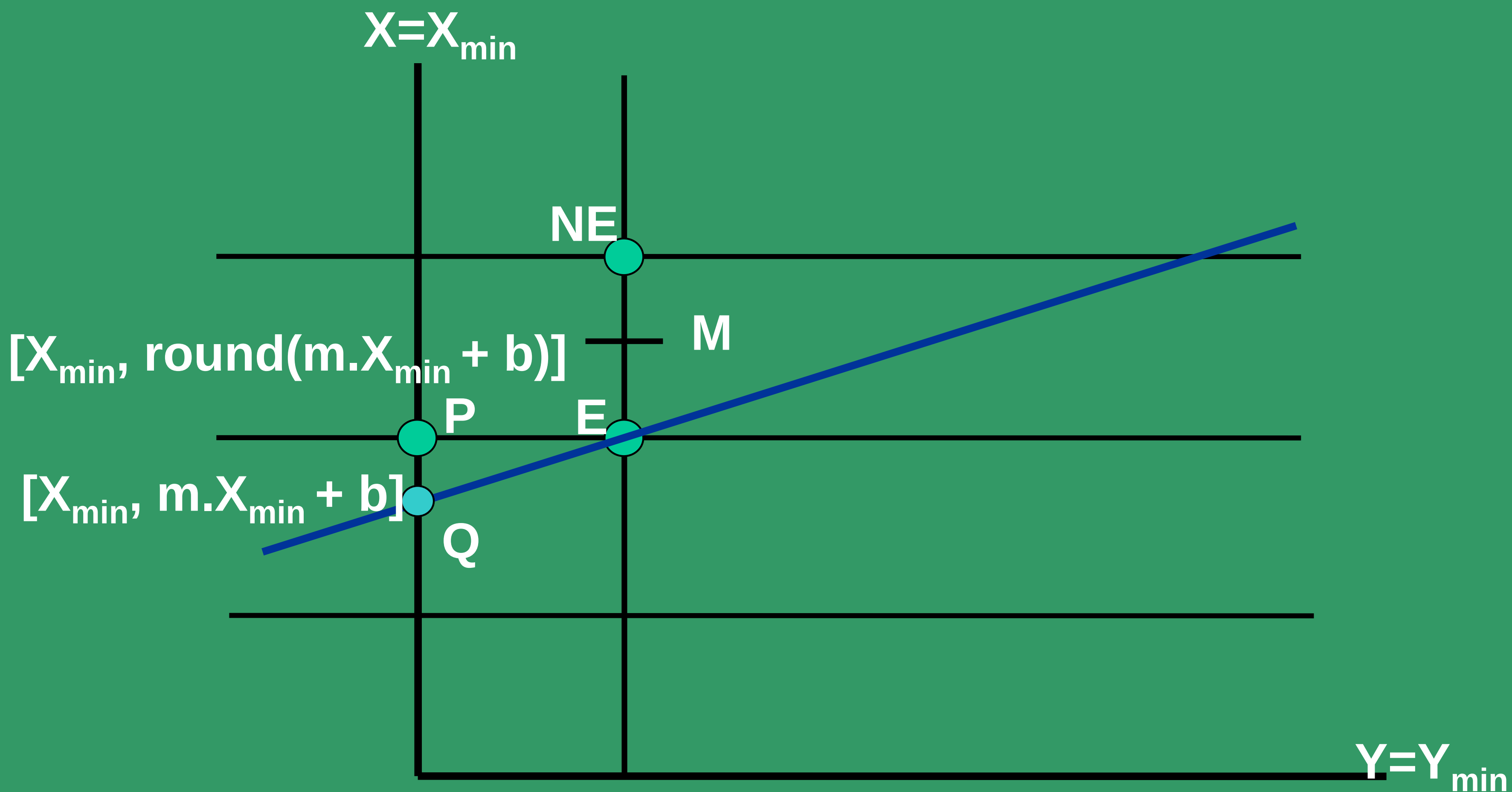
ANTI-ALIASING





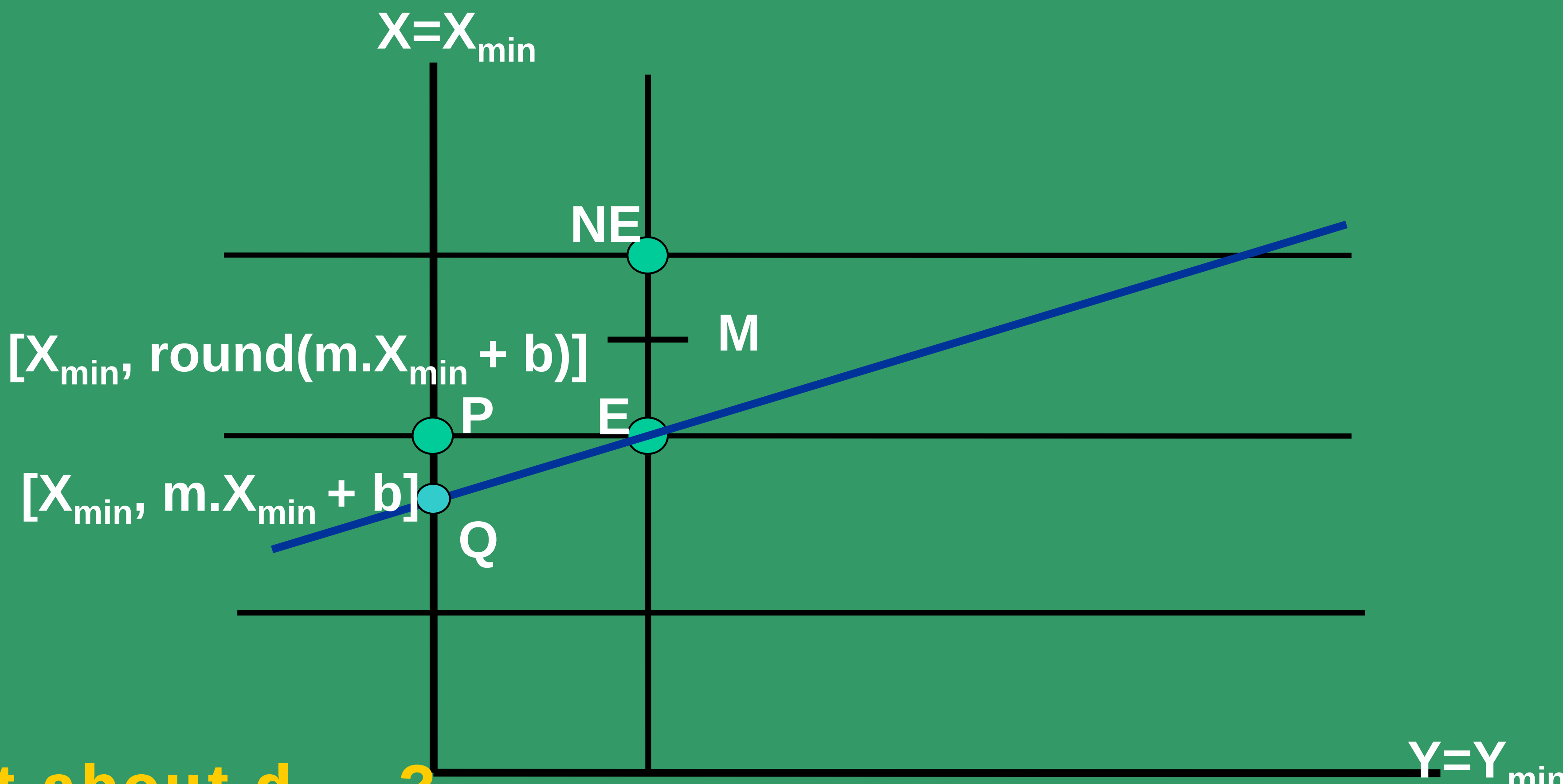


Intersection of a line with a vertical edge of the clip rectangle



No problem in this case to round off the starting point, as that would have been a point selected by mid-point criteria too.

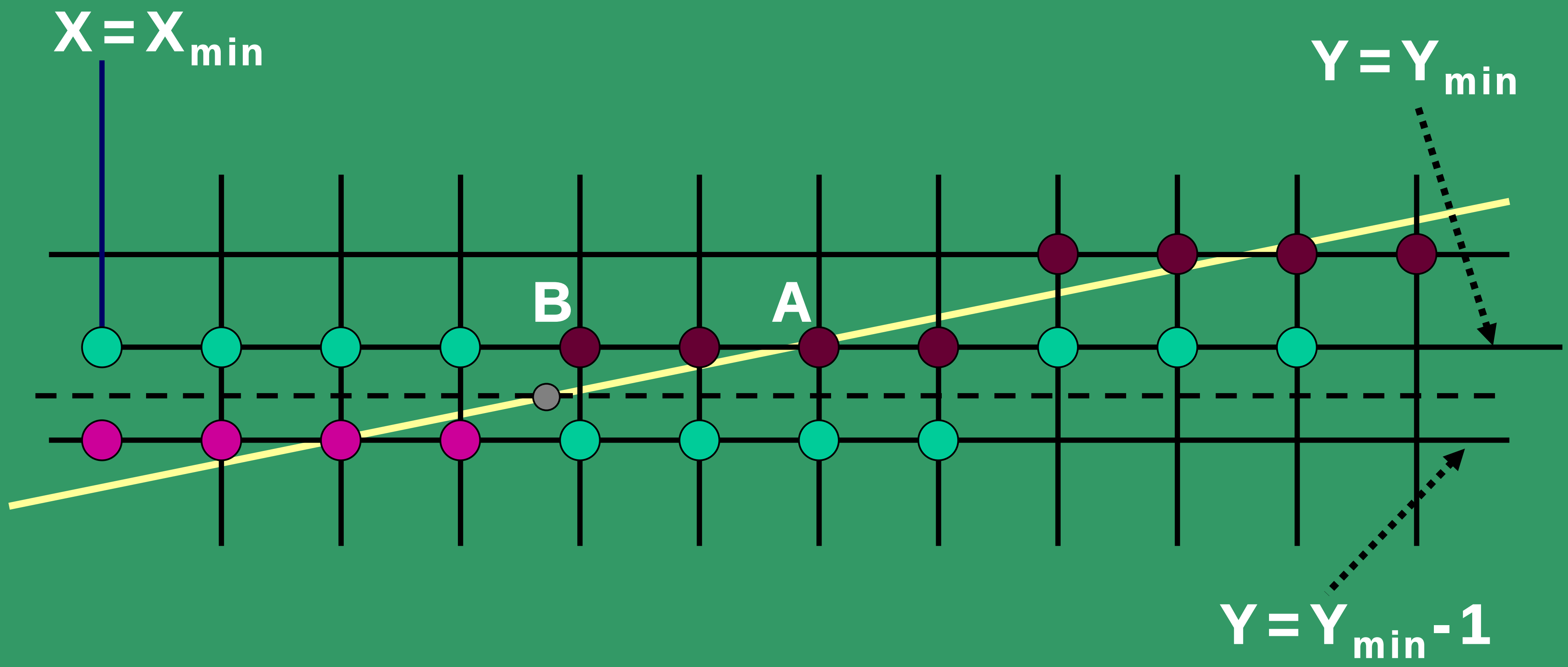
Select P by rounding the intersection point coordinates at Q.



What about d_{start}?

If you initialize the algorithm from P, and then scan convert, you are basically changing "dy" and hence the original slope of the line.

Hence, start by initializing from d(M), the mid-point in the next column, (X_{min} + 1), after clipping).



Intersection of a shallow line with a horizontal edge of the clip rectangle

Intersection of line with edge and then rounding off produces A, not B.

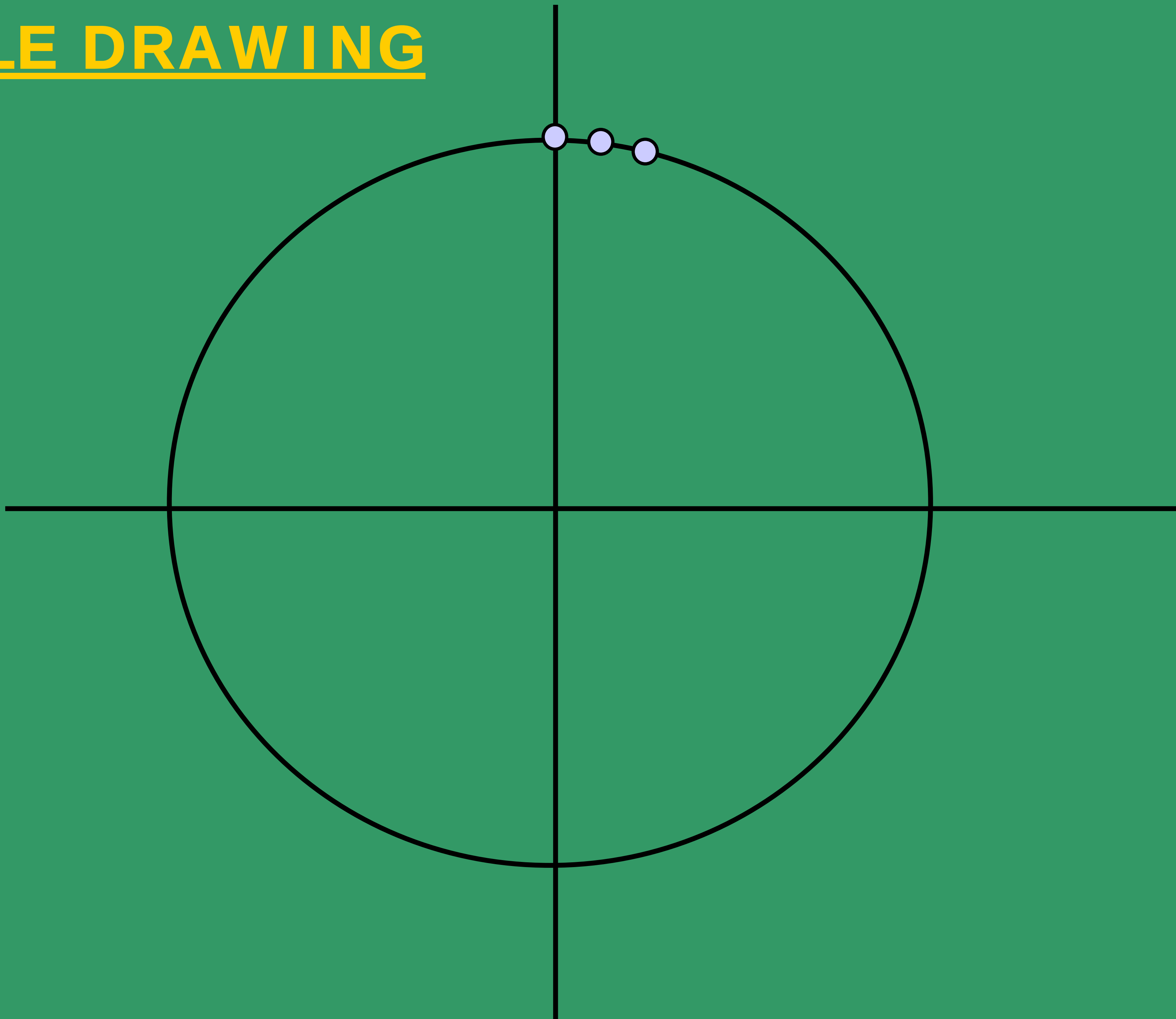
To get B, as a part of the clipped line:

Obtain intersection of line with (Y_{min} - 1/2) and then round off, as

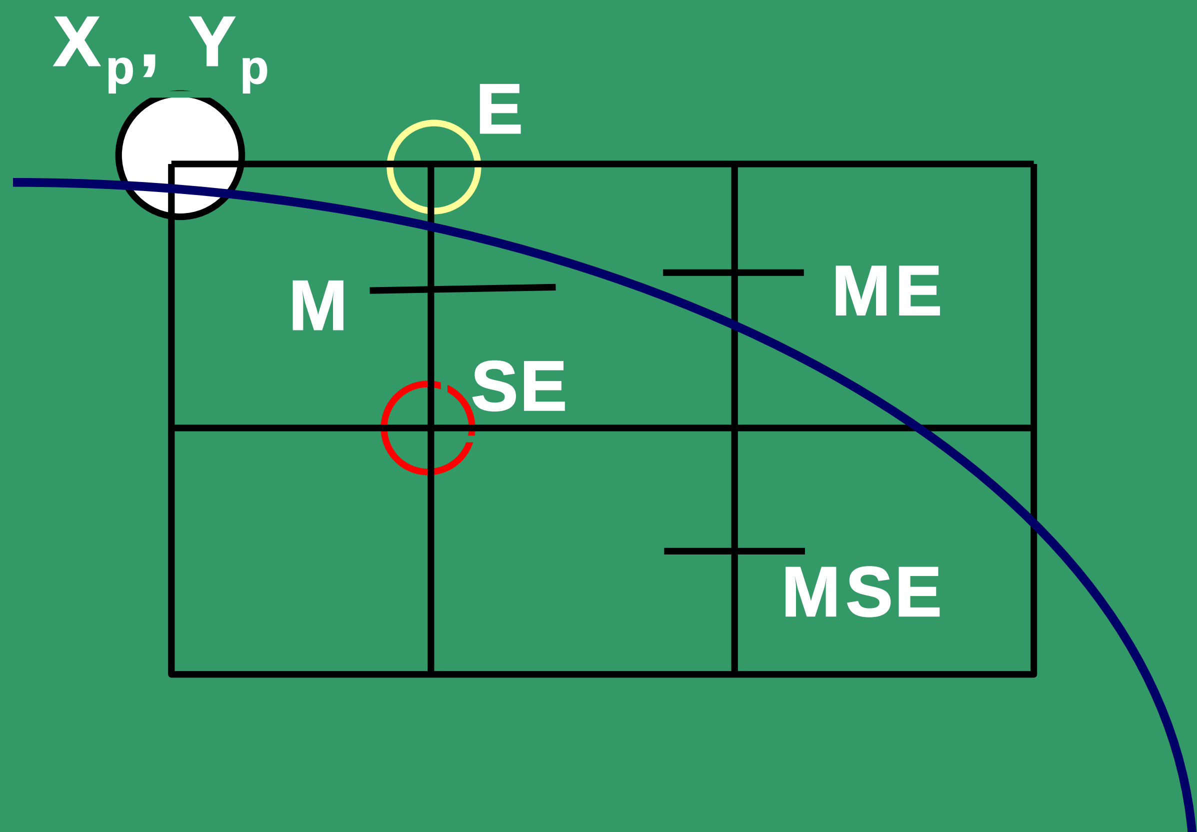
$$B = [\mathrm{round}(X|_{Y_{min}-1/2}), Y_{min}]$$

CIRCLE DRAWING

CIRCLE DRAWING



Assume second octant



Now the choice is between pixels E and SE.

CIRCLE DRAWING

Only considers circles centered at the origin with integer radii.

Can apply translations to get non-origin centered circles.

Explicit equation: $y = +/- \text{sqrt}(R^2 - x^2)$

Implicit equation: $F(x,y)= x^2 + y^2 - R^2 = 0$

Note: Implicit equations used extensively for advanced modeling

(e.g., liquid metal creature from "Terminator 2")

Use of Symmetry: Only need to calculate one octant. One can get points in the other 7 octants as follows:

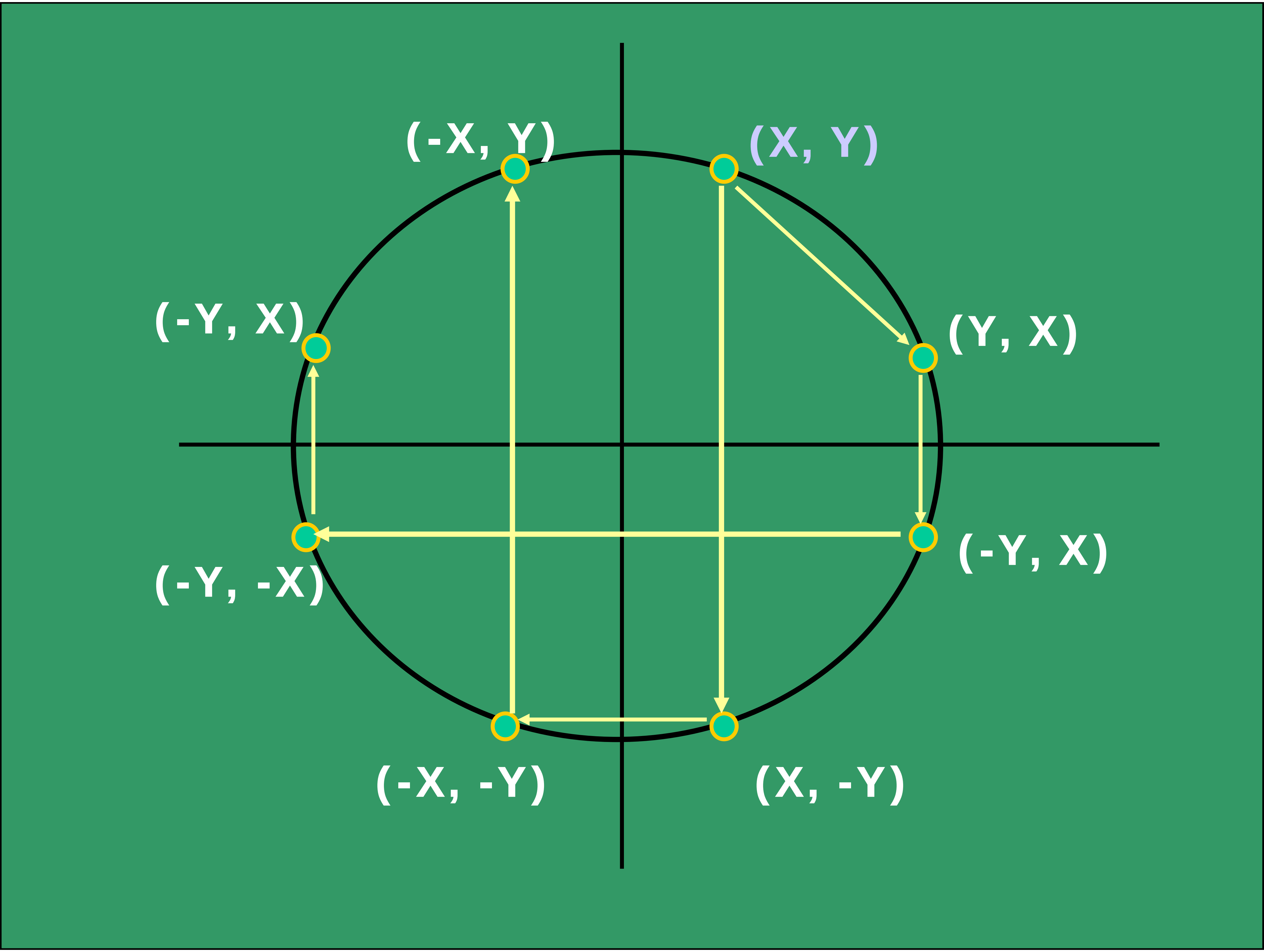
```
Draw_circle(x, y)

begin
    Plotpoint (x, y);  Plotpoint (y, x);

    Plotpoint (x, -y); Plotpoint (-y, x);

    Plotpoint (-x, -y) ; Plotpoint (-y, -x);

    Plotpoint (-x, y); Plotpoint (-y, x);
end
```



(-X, Y)
(X, Y)
(-Y, X)
(Y, X)
(-Y, -X)
(-Y, X)
(-X, -Y)
(X, -Y)

MIDPOINT CIRCLE ALGORITHM

Will calculate points for the second octant.

Use *draw_circle* procedure to calculate the rest.

Now the choice is between pixels E and SE.

$$F(x, y) = x^2 + y^2 - R^2 = 0$$

$F(x, y) > 0$ if point is outside the circle

$F(x, y) < 0$ if point inside the circle.

Again, use $d_{old} = F(M)$;

$$F(M) = F(X_p + 1, Y_p - 1/2)$$
$$= (X_p + 1)^2 + (Y_p - 1/2)^2 - R^2$$

d >= 0 choose SE ; next midpoint: M_{new};

Increment + 1 in X, -1 in y; which gives d_{new}.

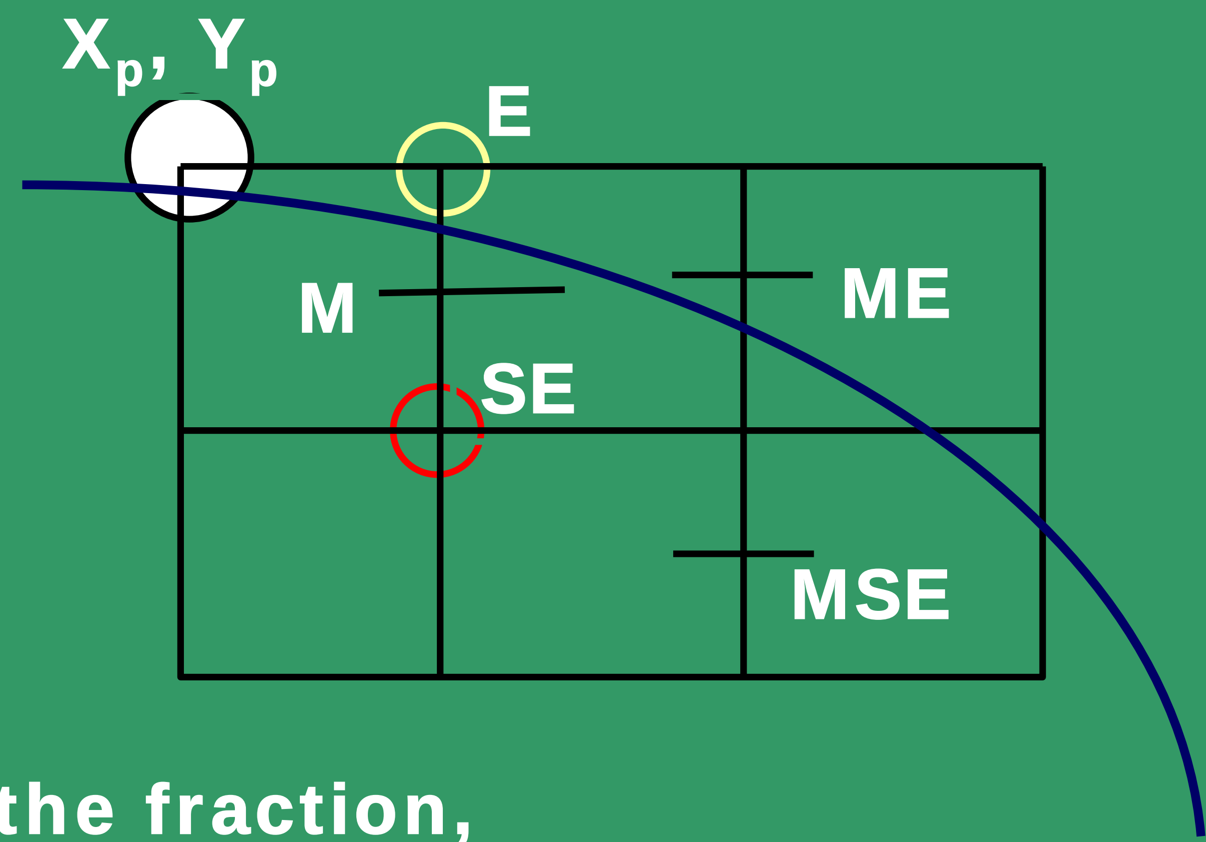
d < 0 choose E ; next midpoint: M_{new};

Increment + 1 in X; which gives = d_{new}.

$$(\Delta d)_{SE} = d_{new} - d_{old}$$
$$= F(X_p + 2, Y_p - 3/2) - F(X_p + 1, Y_p - 1/2)$$
$$= 2X_p - 2Y_p + 5 ;$$

$$(\Delta d)_E = d_{new} - d_{old}$$
$$= F(X_p + 2, Y_p - 1/2) - F(X_p + 1, Y_p - 1/2)$$
$$= 2X_p + 3;$$

$$d_{start} = F(X_0 + 1, Y_0 - 1/2) = F(1, R - 1/2)$$
$$= 1 + (R - 1/2)^2 - R^2 = 1 + R^2 - R + 1/4 - R^2$$
$$= 5/4 - R$$



To get rid of the fraction,

Let $h = d - \frac{1}{4}$ $\Rightarrow$ $h_{start} = 1 - R$

Comparison is: $h < -1/4$.

Since h is initialized to and incremented by integers, so we can just do with: $h < 0$.

The Midpoint Circle algorithm:
(Version 1)

```
x = 0;
y = R;
h = 1 – R;

DrawCircle(x, y);

while (y > x)
    if h < 0   /* select E */
            h = h + 2x + 3;
```

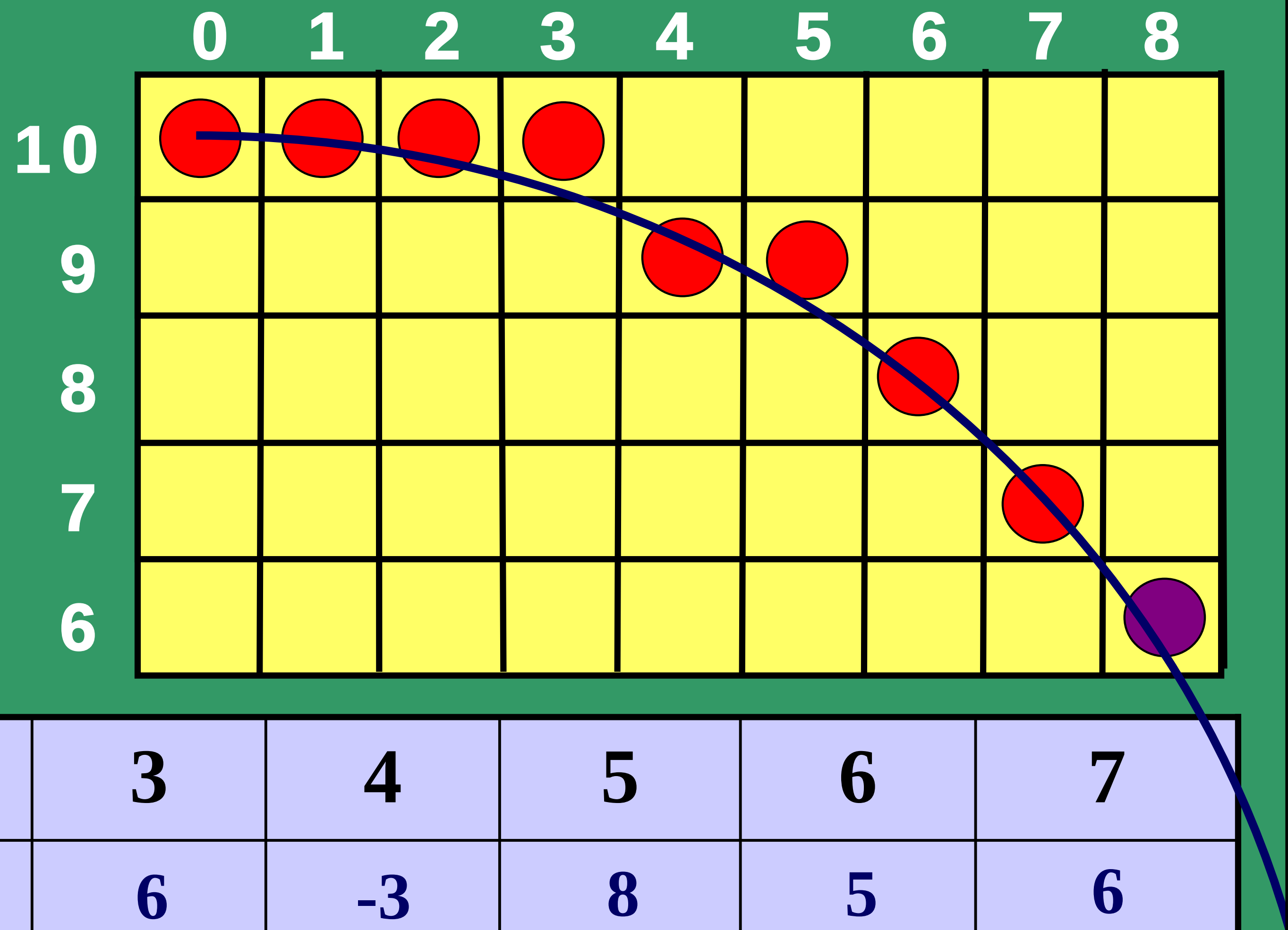
```
        else    /* select SE */
                h = h + 2(x - y) + 5;
                y = y - 1;
    endif

        x = x + 1;
        DrawCircle(x, y);

end_while
```

Example:
R = 10;

Initial Values:
h = 1 – R = -9;
X = 0; Y = 10;
2X = 0;
2Y = 20.



K	1	2	3	4	5	6	7
h	-6	-1	6	-3	8	5	6
2X	0	2	4	6	8	10	12
2Y	20	20	20	20	18	18	16
X, Y	(1, 10)	(2, 10)	(3, 10)	(4, 9)	(5, 9)	(6, 8)	(7, 7)

Problems with this?

Requires at least 1 multiplication and 3 additions per pixel.

Why? Because $(\Delta d)_E$ and $(\Delta d)_{SE}$ are linear functions and not constants.

Solution?

All we have to do is calculate the differences for: $(\Delta d)_E$ and $(\Delta d)_{SE}$ (check if these will be constants). Say, $(\Delta d^2)_E$ and $(\Delta d^2)_{SE}$.

If we chose E, then we calculate $(\Delta d^2)_{E/E}$ and $(\Delta d^2)_{E/SE}$, based on this. Same if we choose SE, then calculate $(\Delta d^2)_{SE/E}$ and $(\Delta d^2)_{SE/SE}$.

If we chose E, go from (X_p, Y_p) to $(X_p + 1, Y_p)$

$$(\Delta d)_{E\text{-}old} = 2X_p + 3, \; (\Delta d)_{E\text{-}new} = 2X_p + 5.$$

Thus $(\Delta d^2)_{E/E} = 2.$

$$(\Delta d)_{SE\text{-}old} = 2X_p - 2Y_p + 5,$$

$$(\Delta d)_{SE\text{-}new} = 2(X_p + 1) - 2Y_p + 5$$

Thus $(\Delta d^2)_{E/SE} = 2.$

If we chose SE,
go from (X_p, Y_p) to $(X_p + 1, Y_p - 1)$

$(\Delta d)_{E\text{-old}} = 2X_p + 3$, $(\Delta d)_{E\text{-new}} = 2X_p + 5$.

Thus $(\Delta d^2)_{SE/E} = 2$.

$(\Delta d)_{SE\text{-old}} = 2X_p - 2Y_p + 5$,

$(\Delta d)_{SE\text{-new}} = 2(X_p + 1) - 2(Y_p - 1) + 5$

Thus $(\Delta d^2)_{SE/SE} = 4$.

So, at each step, we not only increment h, but we also increment $(\Delta d)_E$ and $(\Delta d)_{SE}$.

What are $(\Delta d)_{E\text{-start}}$ and $(\Delta d)_{SE\text{-start}}$?

$(\Delta d)_{E\text{-start}} = 2*(0) + 3 = 3$;

$(\Delta d)_{SE\text{-start}} = 2*(0) - 2*(R) + 5$

The MidPoint Circle Algorithm

(Version 2):

```
x = 0;              y = radius;
h = 1 – R ;
deltaE = 3 ;     deltaSE = -2*R + 5 ;

DrawCircle(x, y);

while (y > x)

        if h < 0      /* select E */

            h = h + deltaE ;

            deltaE  = deltaE + 2;
            deltaSE= deltaSE + 2
```

```
else          /* select SE */

      h = h + deltaSE ;

      deltaE  = deltaE + 2 ;
      deltaSE = deltaSE + 4
     y = y – 1 ;

 endif
x = x + 1 ;

DrawCircle(x, y) ;
end_while
```

Example:

$R = 10;$

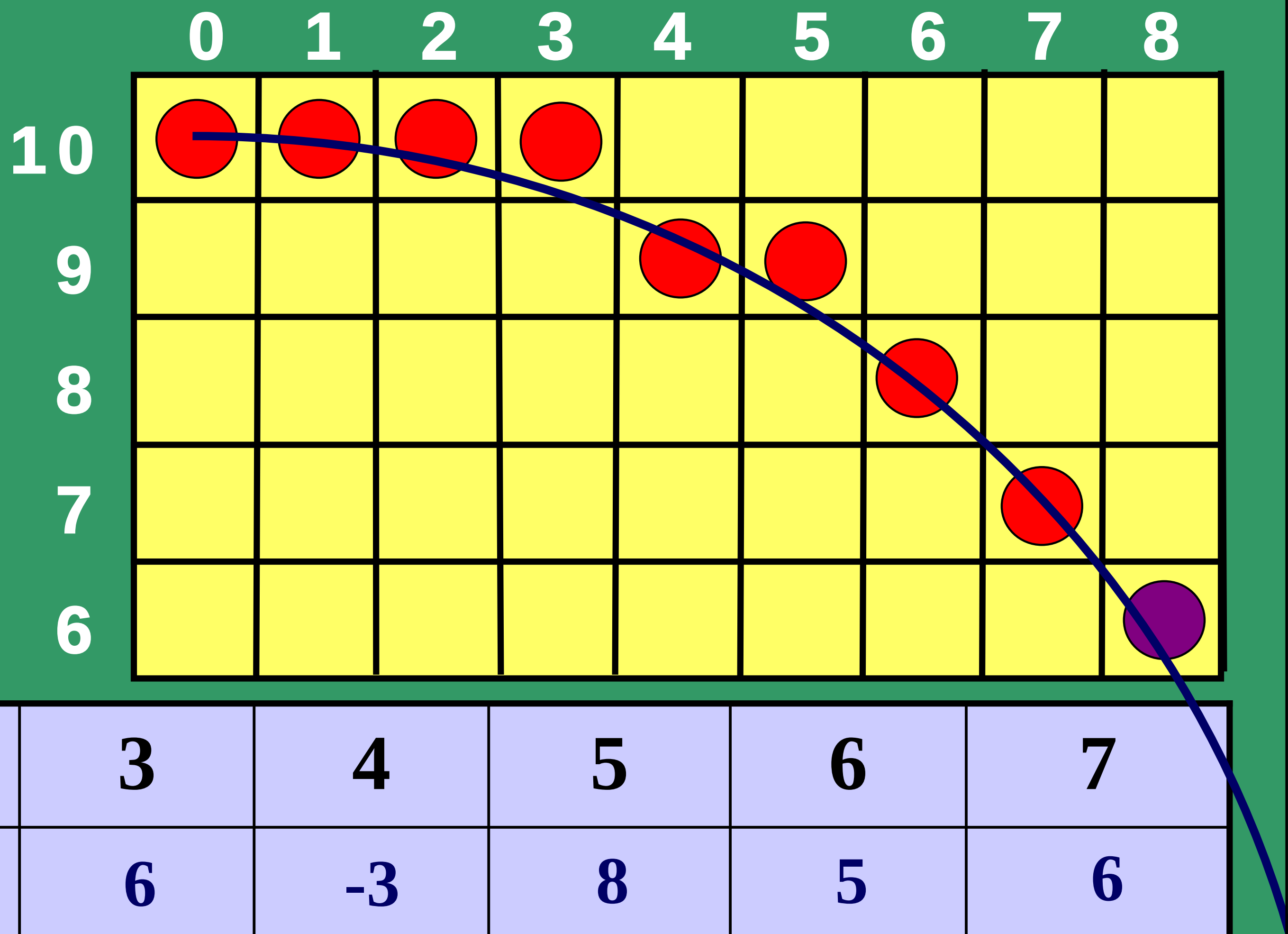
Initial Values:

$X = 0; Y = 10;$

$h = 1 - R = -9;$

$\Delta_E = 3;$

$\Delta_{SE} = -15;$



K	1	2	3	4	5	6	7
h	-6	-1	6	-3	8	5	6
Δ_E	5	7	9	11	13	15	17
Δ_{SE}	-13	-11	-9	-5	-3	1	5
X, Y	(1, 10)	(2, 10)	(3, 10)	(4, 9)	(5, 9)	(6, 8)	(7, 7)

Comparison of the solutions with two different methods

K	1	2	3	4	5	6	7
h	-6	-1	6	-3	8	5	6
Δ_E	5	7	9	11	13	15	17
Δ_{SE}	-13	-11	-9	-5	-3	1	5
X, Y	(1, 10)	(2, 10)	(3, 10)	(4, 9)	(5, 9)	(6, 8)	(7, 7)

K	1	2	3	4	5	6	7
h	-6	-1	6	-3	8	5	6
2X	0	2	4	6	8	10	12
2Y	20	20	20	20	18	18	16
X, Y	(1, 10)	(2, 10)	(3, 10)	(4, 9)	(5, 9)	(6, 8)	(7, 7)