Rasterisation

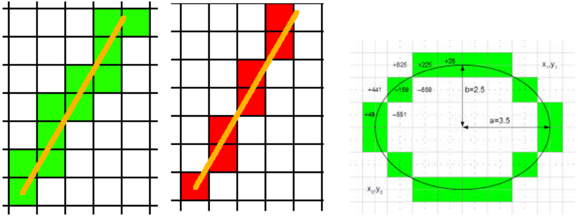
Dr. V Masilamani

masila@iiitdm.ac.in

Department of Computer Science and Engineering
IIITDM Kancheepuram
Chennai-127

Overview

What is Rasterisation

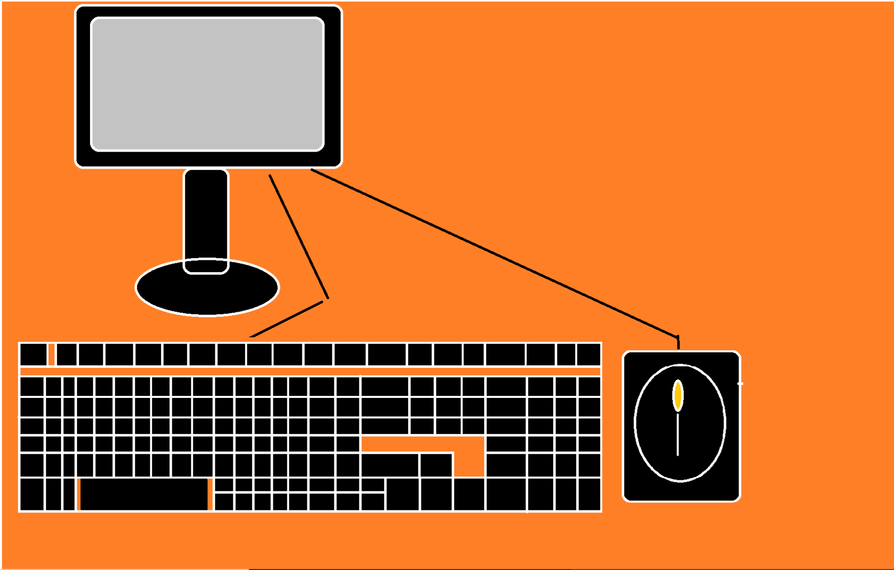


- The process of forming raster(2D-array of pixels) that represents given objects
  - Input: Mathematical representation of objects
  - Output: Digital image representing the objects
  - To draw line segment, the input: two end points of the line segment; output: digital image(raster) representing the line segment

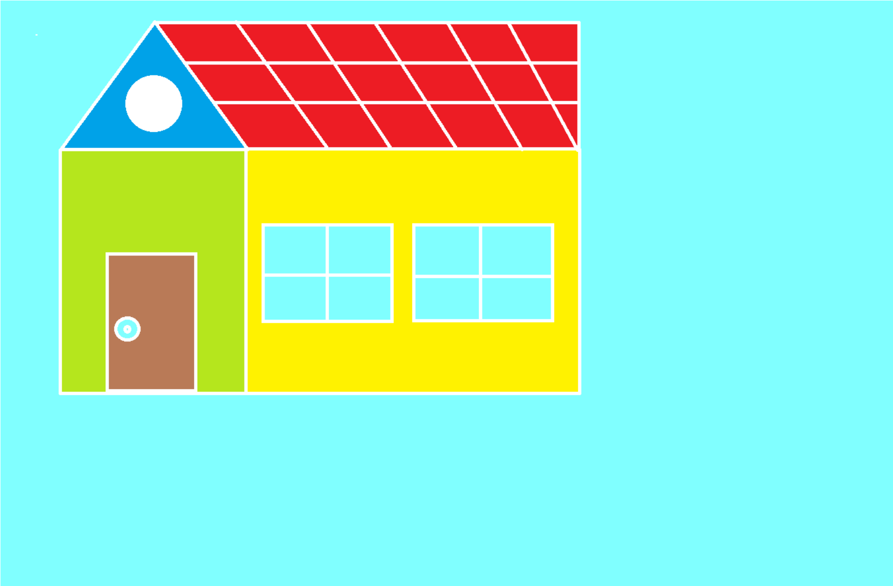
Need of Rasterisation

- Goal of graphics is to synthesise image that represents a scene
- Scene consists of objects of different types
- Objects are in continuous domain in real world
- Sampling is required to represent such object in digital medium
- Sampling is called as **Rasterisation** or **Scan Conversion**
- Scan conversion: scans the object, and converts it into discrete representation

Need of Rasterisation (cont.)

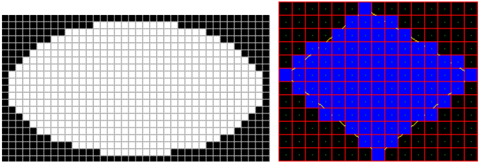


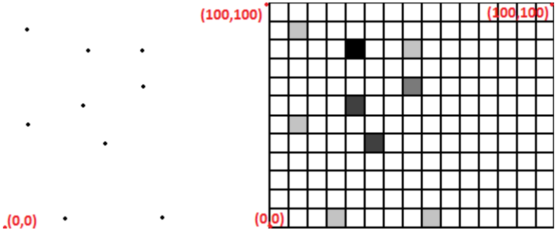
Need of Rasterisation



Types of Rasterisation

- Rasterization of a point
- Rasterization of Boundary of Objects
  - Rasterization of a straight line segment
  - Rasterization of curves such as circle, ellipse etc.
- Rasterization of Region of Objects
  - Rasterization of a interior of ellipse along with boundary
  - Rasterization of interior of polygon along with boundary





- Rasterisation a of point: Given a point (x,y) where $(x, y) \in R \times R$, find $(x', y') \in Z \times Z$ such that (x', y') is very close to (x, y)
- Approach 1: Given $(x, y) \in R \times R$, convert into $x' = \lfloor x \rfloor$, $y' = \lfloor y \rfloor$
- Approach 2: Given $(x, y) \in R \times R$, convert into $x' = \lceil x \rceil$, $y' = \lceil y \rceil$
- Approach 3: Given $(x, y) \in R \times R$, convert into $x' = \lfloor (x + 0.5) \rfloor$, $y' = \lfloor (y + 0.5) \rfloor$ -Round off

Rasterization of straight Line: Given the specification for a straight line, find the collection of locations of pixels(integer coordinates) which closely approximates the line.

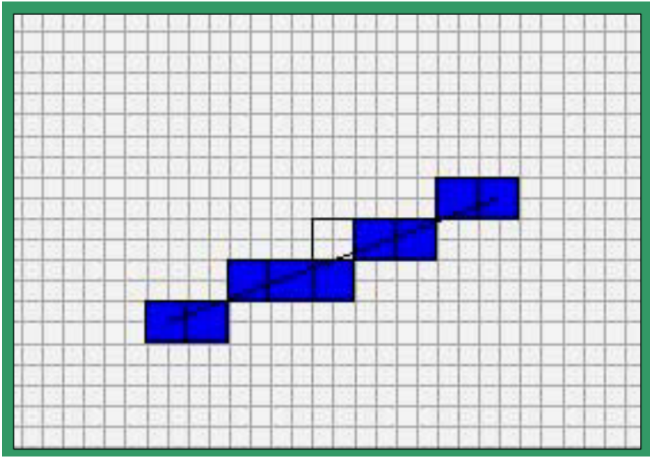
Goals (not all of them are achievable with the discrete space of a raster device):

- 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



Method 1: Direct(Brute Force) Method

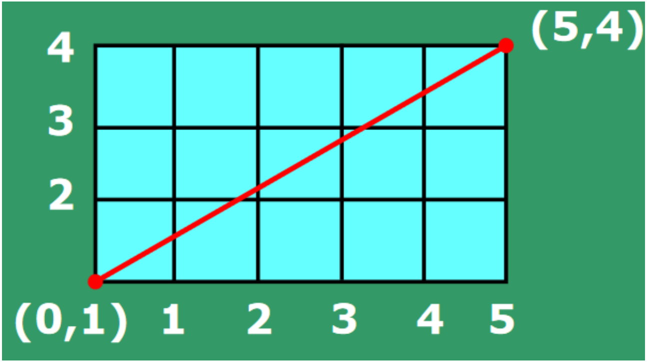
- Given two end points (x_0, y_0) and (x_l, y_l) of the line y=mx+b, find m and b
- Assign values for x from x_0 to x_l, and calculate round(y) from the line equation, and then display the pixel (x, y)

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

Then

- 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)

Ideal Case of a line drawn in a graph paper



Choice of pixels in the raster, as integer values

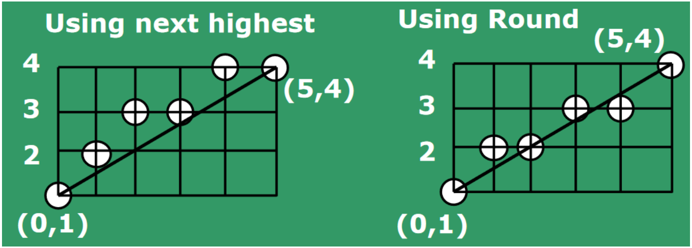
x = 1, y = 2 = round(8/5)

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)



Why is the direct method undesired?

- Floating point multiplication and additions are costly
- Round() function needed -Round() is also costly
- Can get gaps in the line (if slope > 1)

Take another example:

Take another example: y = 10.x + 2

x=1, y=12;

x=2, y=22.

Solution to the gap issue:

- When $m > 1$, Assign values for y from y_0 to y_l, and calculate round(x) from the line equation, and then display the pixel (x, y)

Method 2: DDA - Digital Difference Analyzer

- Let the input to draw the line be two end points (x_0, y_0) and (x_1, y_1)
- The slope of line $y = mx + b$: m=$(y_1 - y_0)$ / $(x_1 - x_0)$
- The first point to be plotted is (x_0, y_0)
- Let (x_i, y_i) be the ith pixel drawn. The next pixel (x_{i+1}, y_{i+1}) can be computed as follows
  Since $y = mx + c$, $y_i = mx_i + b$ and $y_{i+1} = mx_{i+1} + b$
  Therefore $y_{i+1} - y_i = m(x_{i+1} - x_i)$
- Case 1: $|m| \leq 1$
  When $|m| \leq 1$, $x_{i+1} = x_i + 1$
  Therefore $y_{i+1} - y_i = m(x_i + 1 - x_i) = m$ and hence
  $y_{i+1} = y_i + m$ ————–(1)

LINE DRAWING using DDA Algorithm (cont.)

- Case 2: $|m| > 1$
  When $|m| > 1$, $y_{i+1} = y_i + 1$
  Therefore $y_{i+1} - y_i = m(x_{i+1} - x_i)$ will become
  $y_i + 1 - y_i = m(x_{i+1} - x_i)$ and hence
  $x_{i+1} = x_i + 1/m$ ————-(2)
- The equations (1) and (2) provide an incremental algorithm for sampling the line
- Adv: (1) No floating point multiplication; (2) No rounding inside the loop
- Dis Adv: Floating point addition is still there

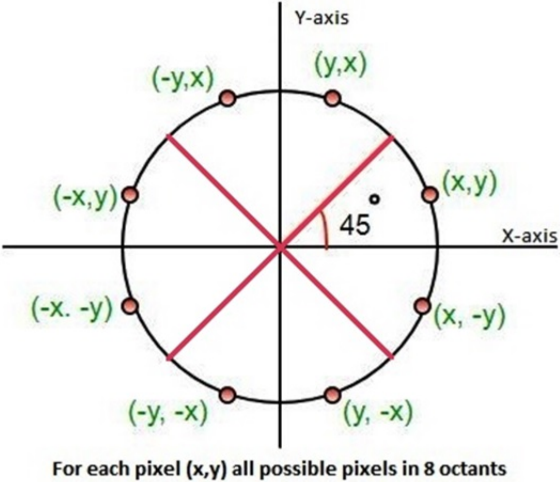
```
void        DDA( ((x0, y0), (x1, y1)) )
{

    //find slope
    float m = (y1 − y0)/(x1 − x0);
    float rm = (x1 − x0)/(y1 − y0);

    //intialization
    int x = round(x0), y = round(y0);
   float yf = y0, xf = x0;


    //Draw first sample
    DrawPoint(x,y)
```

```
        if (|m|<= 1)   //Draw line when |slope|<=1
        {
          while(x < x1)
          {
            x=x+1;     yf=yf+m;
            y=round(yf);
            DrawPoint(x,y)     //display pixel (x,y)
          }
        }
        else //Draw line when |slope|>1
        {
         while(y < y1)
          {
            y=y+1;    xf=x+ rm;
            x=round(xf);
            DrawPoint(x,y)     //display pixel (x,y)
          }
        }
      } //DDA ends
```

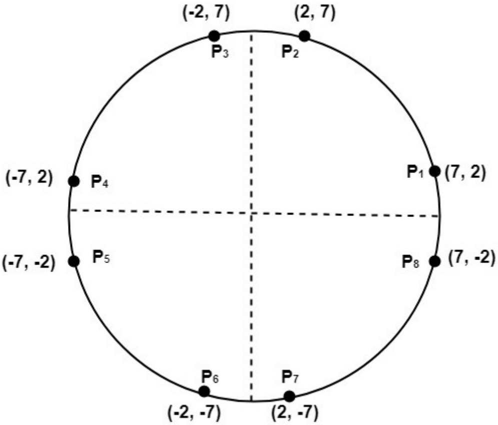
Midpoint Line Drawing Algorithm

- Motivation
  - DDA algorithm was using floating point addition inside the loop, and hence it is slow.
  - Is it possible to get rid of all floating point operations inside the loop?
  - The "yes" answer is provided by the **midpoint line drawing algorithm**
- This algorithm draws line $y = mx + b$, when $0 \leq m \leq 1$, and use symmetry property to draw other lines

Midpoint Line Drawing Algorithm (cont.)

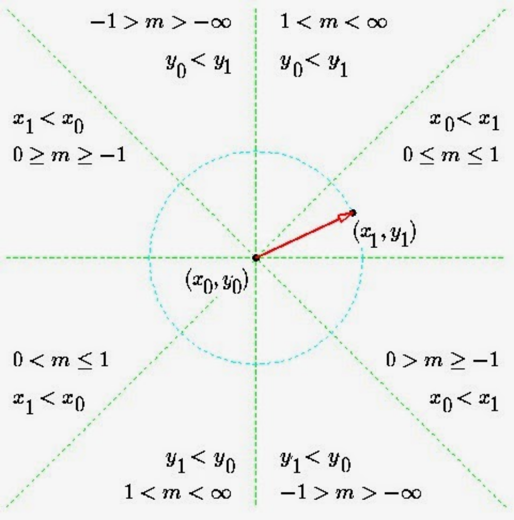


Midpoint Line Drawing Algorithm (cont.)

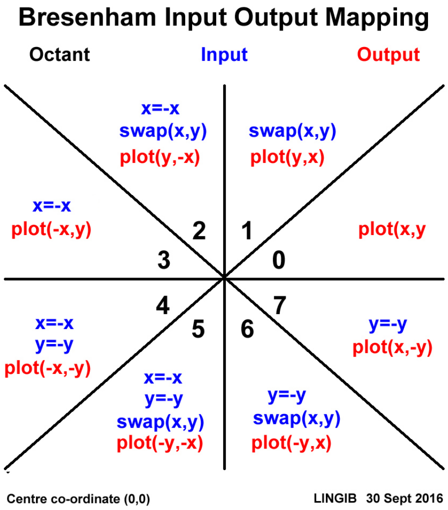


Eight way symmetry of a Circle

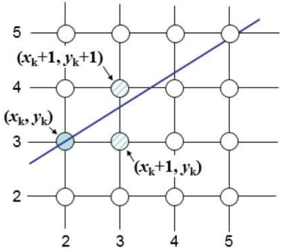
Midpoint Line Drawing Algorithm (cont.)



Midpoint Line Drawing Algorithm (cont.)



Midpoint Line Drawing Algorithm (cont.)



MIDPOINT LINE ALGORITHM

Incremental Algorithm (Assume first octant)

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

Consider the line equations: y=m*x+B

y = (dy/dx) * x + B

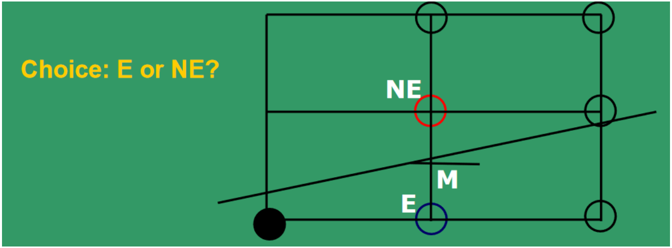
Rewrite as: y*dx=dy*x+dx*B

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

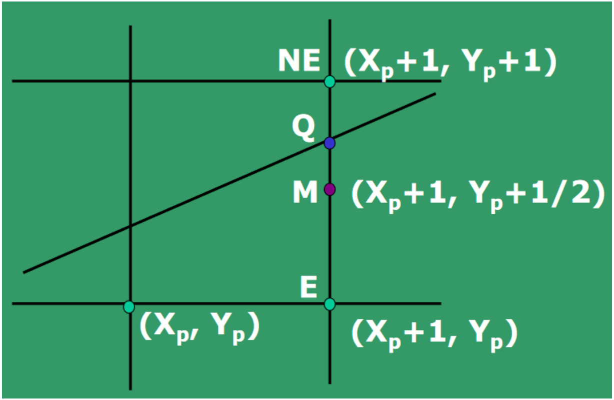
F(x,y) > 0 (ie y< m*x+B); if point (x,y) lies below the line

F(x,y) < 0 (ie y>m*x+B); if point(x,y) lies above the line

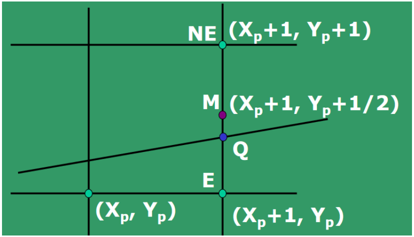
Criteria: Evaluate the mid-point, M, w.r.t. the equation of the line.



If Q is above M, select NE pixel as your next choice (since NE is closer to Q)



If Q is below M, select E pixel as your next choice (since E is closer to Q)



ALGO – for next choice:

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

then Select NE

/*M is below the line*/else

Select E ;

Midpoint Line Drawing Algorithm (cont.)

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

Let (X_p, Y_p) be the pixel computed at iteration p.

Let us compute pixel in the next iteration (X_{p+1}, Y_{p+1}) in terms of (X_p, Y_p)

Let us define a decision variable $d_p = F(M_p)$,
where M_p is the mid point at the iteration p,
and given as $M_p = (x_{p+1}, (y_p + y_p + 1)/2) = (x_p + 1, y_p + 1/2)$

$d_p = \mathrm{F}(X_p{+}1, Y_p{+}1/2) = \mathrm{dy}(X_p{+}1)\text{-}\mathrm{dx}(Y_p{+}1/2){+}\mathrm{B}\ \mathrm{dx}$,

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

ie if $d_p < 0$ (Mp is above the line) then the next pixel is E: $(x_p + 1, y_p)$

if $d_p >= 0$ (Mp is below the line) then the next pixel is NE: $(x_p + 1, y_p + 1)$

Case I. Chosen E at iteration (p): The next pixel we get is $(x_{p+1}, y_{p+1}) = (x_p + 1, y_p)$

Hence in the next iteration $p + 1$, the mid point E and NE of $(x_p + 1, y_p)$ will be

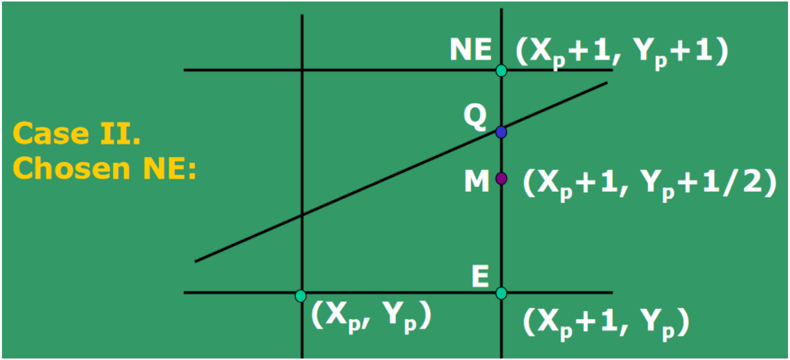
$M_{p+1} = (x_p + 2, (y_p + y_p + 1)/2) = (x_p + 2, y_p + 1/2)$

$d_{p+1} = F(M_{p+1}) = \mathrm{F}(X_p+2, Y_p+1/2) = \mathrm{dy}(X_p+2) + \text{-}\mathrm{dx}(Y_p+1/2) + \mathrm{B}$ dx

$(\Delta d)_E = d_{p+1}$ -$d_p =$ dy

$$d_{p+1} = d_p + dy$$

$$d_{p+1} = d_p + (\Delta d)_E$$



Case II. Chosen NE at iteration (p): The next pixel we get is $(x_{p+1}, y_{p+1}) = (x_p + 1, y_p + 1)$

Hence in the next iteration $p + 1$, the mid point E and NE of $(x_p + 1, y_p + 1)$ Will be
$M_{p+1} = (x_p + 2, (y_{p+1} + y_{p+2})/2) = (x_p + 2, y_p + 3/2)$

Midpoint Line Drawing Algorithm (cont.)

d_{p+1}= $F(M_{p+1})$ = F(X_p+2, Y_p+3/2) = dy(X_p+2) + -dx(Y_p+3/2) + B dx

$(\Delta d)_{NE}$ = d_{p+1} -d_p= /* = dy - dx */

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

Update using d_{p+1} = d_p+ Δd_{NE}

Midpoint criteria

At iteration p

$d_p = F(M_p) = \mathrm{F}(X_p{+}1,\ Y_p{+}1/2);$

if $d_p > 0$ choose NE

else /* if $d_p \leq 0$ */choose E ;

Case EAST :

- $d_{p+1} = \mathrm{F}(M_{p+1}) = \mathrm{F}(X_p + 2,\ \mathrm{Y} + 1/2)$
- d_{p+1} -d_p= dy
- $d_{p+1} = d_p +$ dy
- Let $(\Delta d)_E =$ dy
- $d_{p+1} = d_p + (\Delta d)_E$

Case NORTH-EAST :

- $d_{p+1} = F(M_{p+1}) = F(X_p + 2,\ Y_p + 3/2)$
- d_{p+1}-d_p = dy - dx
- Let $(\Delta d)_{NE}$ = dy - dx
- $d_{p+1} = d_p + (\Delta d)_{NE}$

We have obtained iterative definition of d_p to chose E on NE at iteration p

What is d_0?

As the starting pixel is (x_0, y_0),
$M_0 = (x_0 + 1, (y_0 + y_0 + 1)/2) = (x_0 + 1, y_0 + 1/2)$

$d_0 = F(M_0) = \mathrm{F}(x_0 + 1,\ y_0 + 1/2)$
$= dy(x_0 + 1) - dx(y_0 + 1/2) + B * dx$
$=(dy * x_0 - dx * y_0 + B * dx) + (dy - dx/2) = \mathrm{F}(x_0,\ y_0)+$
$(dy - dx/2)$
$=$ 0+ dy - dx/2 (Assuming end points are integer coordinates)
$=dy - dx/2$

Midpoint Line Drawing Algorithm (cont.)

Let's get rid of the fraction and see what we end up with for all the variables:
When d_0 is re-assigned as $d_0 = 2d_0$,
$d_0 = $ 2dy – dx ;
$(\Delta d)_E =$ 2dy ; (when $d_{p+1} = d_p + (\Delta d)_E$)
$(\Delta d)_{NE} =$ 2(dy - dx) ;(when $d_{p+1} = d_p + (\Delta d)_{NE}$)
Note: $d_p > or < or = 0$ iff $2d_p > or < or = 0$ respectively

The Midpoint Line Algorithm

// **Initialization:**

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) ;

//display the first point PlotPoint(x,y)

Midpoint Line Drawing Algorithm (cont.)

The Midpoint Line Algorithm(contd.)

//Plot the remaining points to display line

while $x < x_1$ **do**
    if $d \leq 0$ **then** ▷ /* Choose E */
        d = d + $(\Delta d)_E$;
    else ▷ /* Choose NE */
        d = d + $(\Delta d)_{NE}$;
        y = y + 1
    end if
    x = x + 1 ;
    PlotPoint(x, y) ;
end while

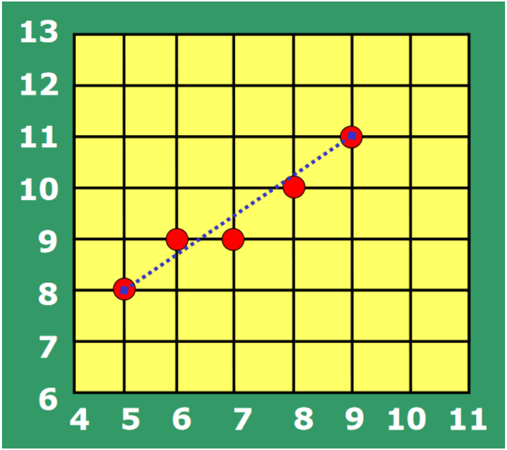
Midpoint Line Drawing Algorithm (cont.)

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)

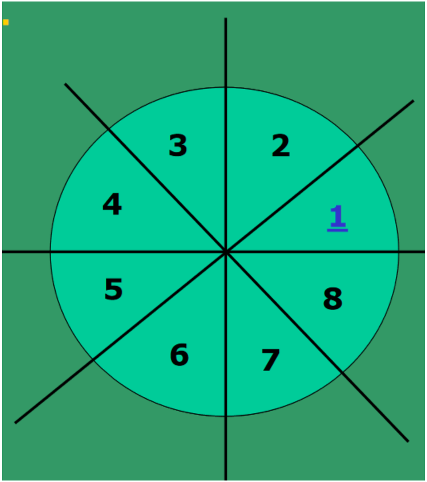
INIT: dy = 3; dx = 4; d_0=2; $(\Delta d)_E$ = 6; $(\Delta d)_{NE}$ = -2;

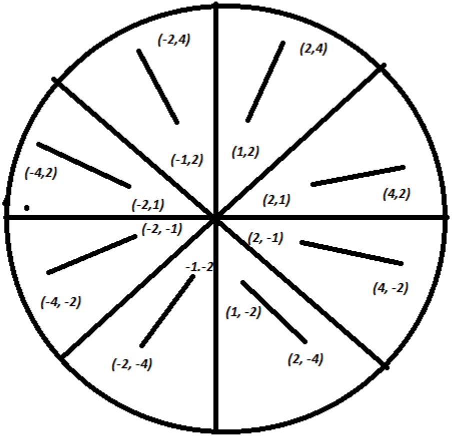
Midpoint Line Drawing Algorithm (cont.)



Midpoint Line Drawing Algorithm (cont.)

We have considered lines in the first octant only. What about rest?




(-2,4)
(2,4)
(-1,2)
(1,2)
(-4,2)
(4,2)
(-2,1)
(2,1)
(-2, -1)
(2, -1)
-1.-2
(-4, -2)
(4, -2)
(1, -2)
(-2, -4)
(2, -4)

Midpoint Line Drawing Algorithm

- Octant 2: If a line is from (x_1, y_1) to $(x_2, y2)$ with $1 \leq m \leq \infty$ (theta is in the range 45 degree to 90 degree), then
  - (Mirror diagonally) Transformed it to line from (y_1, x_1) to (y_2, x_2) with $0 \leq m \leq 1$
  - Find pixel locations (x_i, y_i) for all i for the transformed line
  - Draw pixels (y_i, x_i) for all i
- Octant 3: If a line is from (x_1, y_1) to $(x_2, y2)$ with $11 \leq m \leq -\infty$ (angle is in the range 90 to 135) then
  - Mirror Vertically
  - Mirror diagonally
  - The transformed line will be with $0 \leq m \leq 1$
  - find (x_i, y_i) for all i
  - DrawPoints after reversing operations: ie mirror diagonally and then mirror vertically

Midpoint Line Drawing Algorithm (cont.)

- Octant 4: If a line is from (x_1, y_1) to $(x_2, y2)$ with $-1 \leq m \leq 0$ (angle is in the range 135 deg to 180 deg) then
  - (Mirror Vertically) Transformed it to line from $(-x_1, y_1)$ to $(-x_2, y_2)$ with $0 \leq m \leq 1$
  - Find pixel locations (x_i, y_i) for all i for the transformed line
  - Draw pixels $(-x_i, y_i)$ for all i
- Octant 5: If the angle is in the range 180 deg to 225 deg, then
  - Replace x by -x and Replace y by -y
  - Find pixel locations (x_i, y_i) for all i for the transformed line (line in Octant 1)
  - Draw pixels $(-x_i, -y_i)$ for all i
- Octant 6: If the angle is in the range 225 deg to 270 deg, then
  - Replace x by -x and Replace y by -y (Results in Octant 2)

Midpoint Line Drawing Algorithm (cont.)

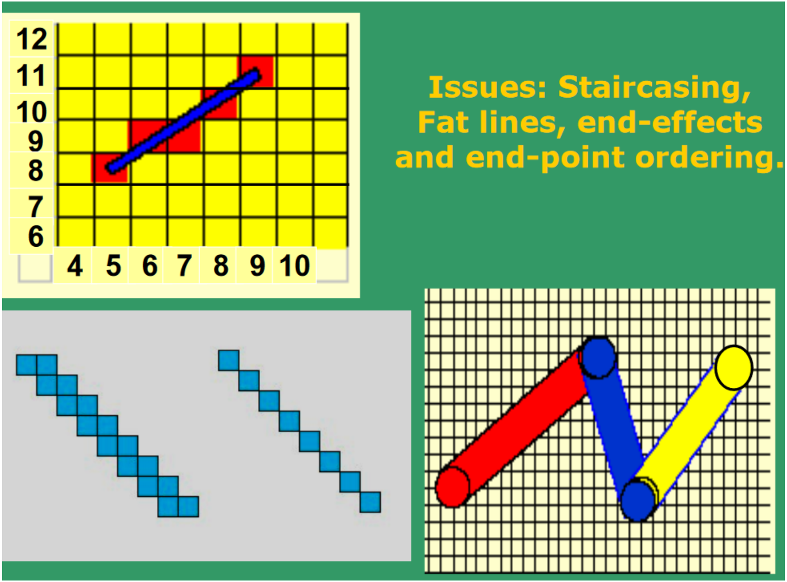
  - Find pixel locations (x_i, y_i) for all i for the transformed line (line in Octant 2)
  - Draw pixels $(-x_i, -y_i)$ for all i
- Octant 7: If the angle is in the range 270 deg to 315 deg, then
  - Replace x by -x and Replace y by -y (Results in Octant 3)
  - Find pixel locations (x_i, y_i) for all i for the transformed line (line in Octant 3)
  - Draw pixels $(-x_i, -y_i)$ for all i
- Octant 8: If the angle is in the range 315 deg to 360 deg, then
  - Replace x by -x and Replace y by -y (Results in Octant 4)
  - Find pixel locations (x_i, y_i) for all i for the transformed line (line in Octant 4)
  - Draw pixels $(-x_i, -y_i)$ for all i

Midpoint Line Drawing Algorithm (cont.)

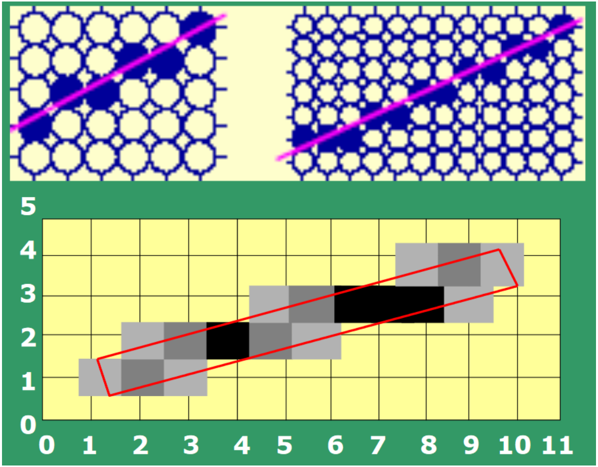
Bresenham Line Drawing Algorithm Vs Mid-point Line Drawing Algorithm

- Criteria Used in Mid point algorithm:
  - Select E if mid point of E and NE is above the line
  - Select NE if mid point of E and NE is below the line
- Criteria Used in Bresenham algorithm:
  - Select E if d(E, Line) $\leq$ d(NE, Line)
  - Select NE if d(E, Line) $>$ d(NE, Line)
- Both lead to the same updates to get next pixel (x_{i+1}, y_{i+1}) from (x_i, y_i)

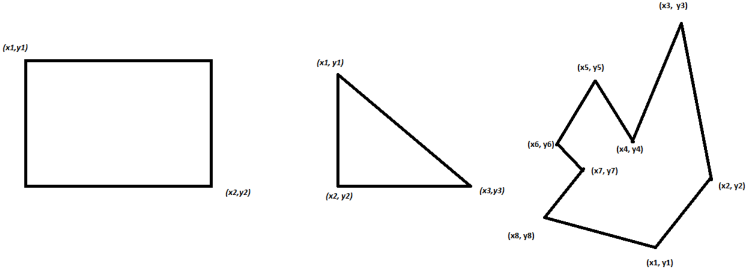
Midpoint Line Drawing Algorithm (cont.)



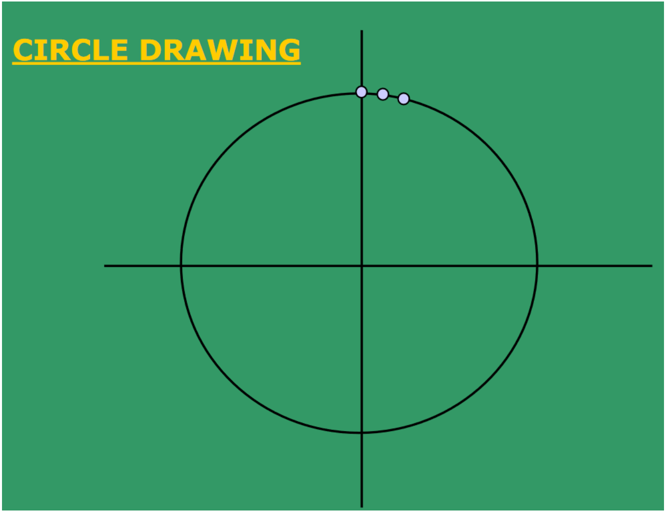
ANTI-ALIASING



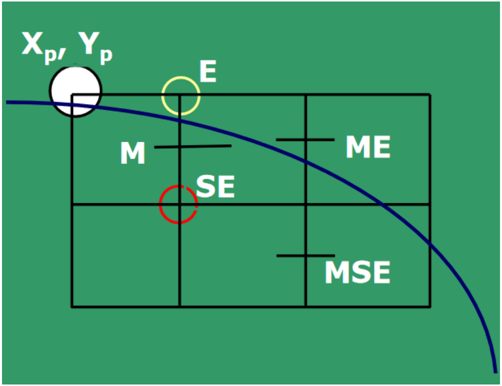
Midpoint Line Drawing Algorithm (cont.)



Some Applications of Line Drawing: Drawing boundary of polygons and also filling polygons


CIRCLE DRAWING

Consider second octant



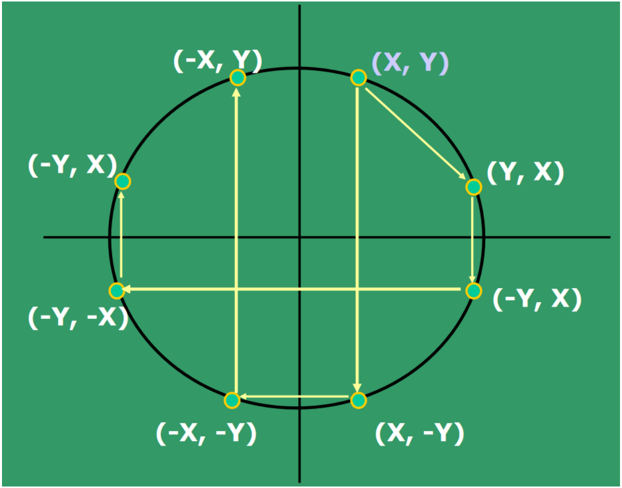
Now the choice is between pixels E and SE.

CIRCLE DRAWING

- Consider circles centered at the origin with integer radii.
- Translations can be applied to get non-origin centered circles.
- Explicit equation: y = +/- sqrt(R^2-x^2)
- Implicit equation: F(x,y)= x^2+ y^2-R^2=0
- Implicit equations used extensively for advanced modeling (e.g., liquid metal creature from "Terminator 2") Symmetry of points in Circle: Rasterizing one point will produce rasterisation of 7 more points

RASTERIZATION OF CIRCLE / CIRCLE DRAWING (cont.)

Symmetry of points in Circle: Rasterizing one point will produce rasterisation of 7 more points



Use of Symmetry: Rasterize second octant. Rasterize other 7 octants as follows:

for each $(x, y) \in 2^{nd} Octant$ DrawCircle(x, y)

DrawCircle(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

MIDPOINT CIRCLE ALGORITHM

- What is special about 2^{nd} octant
  - The first point to be drawn is $(0, r)$ // assume r is integer
  - Since tangent at $(0, r)$ is horizontal and tangent at (x, y) when $x = y$ is diagonal at 135 degree, movement needs to either horizontal(E) or diagonal at 135 degree (SE)
  - After drawing (x_i, y_i), to draw (x_{i+1}, y_{i+1}), E or SE point of (x_i, y_i) needs to be chosen
- How to decide between E and SE
  - Circle Equation: 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.
  - At iteration p, Let M_p be the mid point of E and SE

RASTERIZATION OF CIRCLE / CIRCLE DRAWING (cont.)

- $d_p = F(M_p)$, where
  $M_p = (x_p + 1, (y_p + y_p - 1)/2) = (x_p + 1, y_p - 1/2)$;
- $d_p = F(M_p) = \mathrm{F}(x_p + 1,\ y_p - 1/2) = (x_p + 1)^2 + (y_p - 1/2)^2 - R^2$
- Mid Point Criteria: Choose SE when $d_p \geq 0$; Choose E when $d_p < 0$

Let us derive iterative definition for d_{p+1}

d_{p+1} is dependent on d_p

$Case: \; d_p \geq 0$:
SE is chosen at iteration p. Therefore $x_{p+1} = x_p + 1$ and
$y_{p+1} = y_p - 1$

next midpoint: M_{p+1} will be mid point of E and SE of
$(x_p + 1, y_p - 1)$
which is $(x_p + 2, ((y_p - 1) + (y_p - 2))/2) = (x_p + 2, y_p - 3/2)$

$d_{p+1} = F(M_{p+1}) = (x_p + 2)^2 + (y_p - 3/2)^- R^2$
$d_{p+1} - d_p = 2x_p 2Y_p + 5$
Let $(\Delta d)_{SE} = d_{p+1} - d_p$
Hence $d_{p+1} = d_p + (\Delta d)_{SE}$, where $(\Delta d)_{SE} = 2x_p - 2y_p + 5$

$Case: \; d_p < 0$:
E is chosen at iteration p. Therefore $x_{p+1} = x_p + 1$ and $y_{p+1} = y_p$

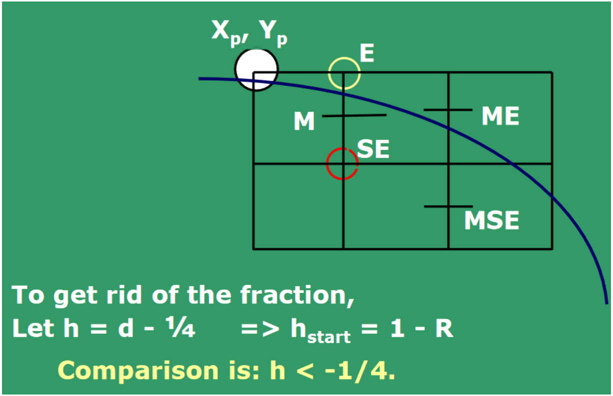
next midpoint: M_{p+1} will be mid point of E and SE of $(x_p + 1, y_p)$

which is $(x_p + 2, ((y_p) + (y_p - 1))/2) = (x_p + 2, y_p - 1/2)$

$d_{p+1} = F(M_{p+1}) = (x_p + 2)^2 + (y_p - 1/2)^{-} R^2$
$d_{p+1} - d_p = 2X_p + 3$
Let $(\Delta d)_E = d_{p+1} - d_p$
Hence $d_{p+1} = d_p + (\Delta d)_E$, where $(\Delta d)_E = 2x_p + 3$

$d_{start} = d_1 = F(M_1)$, where M_1 mid point of E and SE of $(0, R)$
$= F((1+1)/2, (R + R - 1)/2) =$ F(1, R -1/2)
$=$1 + $(R - 1/2)^2$-R^2= 1 + R^2- R + 1/4 - R^2
$=$5/4 - R

RASTERIZATION OF CIRCLE / CIRCLE DRAWING (cont.)



Summary:

- $d_{p+1} = d_p + (\Delta d)_{SE}$, where $(\Delta d)_{SE} = 2X_p - 2Y_p + 5$ when $d_p >= 0$
- $d_{p+1} = d_p + (\Delta d)_E$, where $(\Delta d)_E = 2X_p + 3$ when $d_p < 0$
- $d_1 = 5/4 - R$, which is approximated as $d_1 = 1 - R$

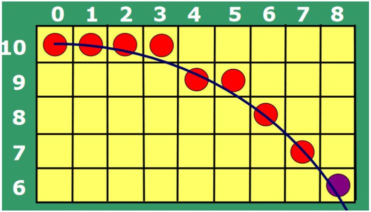
RASTERIZATION OF CIRCLE / CIRCLE DRAWING (cont.)

The Midpoint Circle algorithm

```
x = 0
y = R
d = 1 - R
DrawCircle(x, y);
while y > x do
    if d < 0 then                          ▷ /* Choose E */
        d = d + 2x + 3;
    else                                   ▷ /* Choose SE */
        d = d + 2(x - y) + 5;;
        y = y - 1
    end if
    x = x + 1 ;
    DrawCircle(x, y) ;
end while
```

Example: R = 10; Initial Values:d = 1 − R = -9; X = 0; Y = 10; 2X = 0; 2Y = 20.

p	**2**	**3**	**4**	**5**	**6**	**7**	**8**
d	-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)

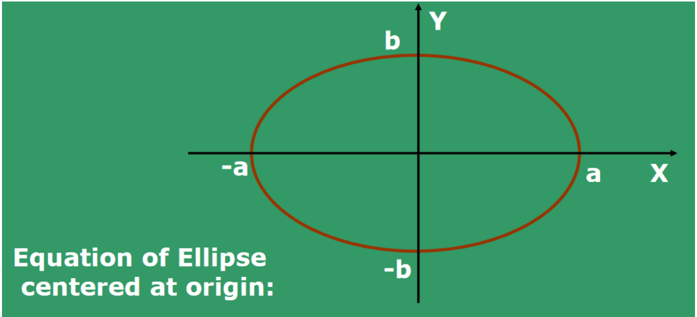


RASTERIZATION OF CIRCLE / CIRCLE DRAWING (cont.)

Observation:

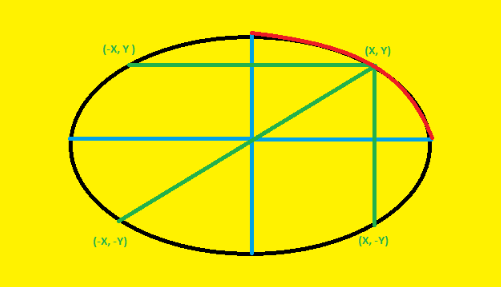
- No floating point operation in side the loop
- Only integer addition, decrement, increment and shifting left (multiplication by 2) are used
- Time complexity is $O(n)$, where n is number of points to be drawn

SCAN CONVERTING ELLIPSES



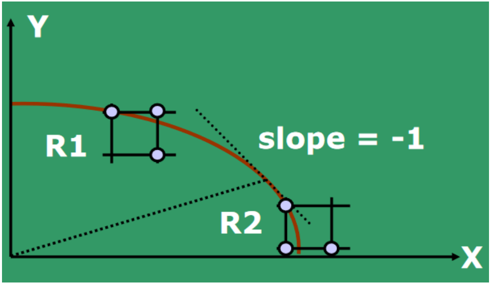
$$F(X,Y) = b^2X^2 + a^2Y^2 - a^2b^2 = 0$$

Length of the major axis: 2a; and minor axis: 2b.



- Scheme: Rasterise and draw first quadrant, and then draw other quadrants using symmetry
  Let (x, y) be in first quadrant
  - $(-x, y)$ will be in the second quadrant
  - $(-x, -y)$ will be in the third quadrant
  - $(x, -y)$ will be in the fourth quadrant

RASTERIZATION OF ELLIPSE / ELLIPSE DRAWING (cont.)



- Split the the first quadrant into two regions R1 and R2, where R1 consists of all points on ellipse from $(0, b)$ to (x^*, y^*) s.t slope at (x^*, y^*) is -1
- Properties of points in R1: Angle of tangent at $(0, b)$ is 180 degree (E) angle of tangent at (x^*, y^*) is 135 degree(SE).
  Therefore to move from p^{th} point to $(p+1)^{th}$ point, go to E or go to SE

RASTERIZATION OF ELLIPSE / ELLIPSE DRAWING (cont.)

- Properties of points in R2: angle of tangent at (x^*, y^*) is 135 degree(SE) and angle of tangent at $(a, 0)$ is 90 degree(S). Therefore to move from p^{th} point to $(p+1)^{th}$ point, go to SE or go to S
- The choice of pixels in R1 is between E and SE, whereas in R2, the choice is between S and SE.
- We need to obtain the point on the contour where the slope of the curve is -1.

F(X,Y) = b^2X^2 + a^2Y^2 - a^2b^2

The boundary point between R1 and R2 is (x, y) with slope = -1.

Hence at (x, y), dY/dX = -1

WKT When F(X, Y)=0, dY/dX = - $\frac{\partial F}{\partial X} / \frac{\partial F}{\partial Y}$

The slope of Ellipse from $(0, b)$ to $(a, 0)$ monotonically decreases from 0 to $-\infty$

Therefore $\frac{\partial F}{\partial X} / \frac{\partial F}{\partial Y}$ increases monotonically from 0 to ∞

For each point in R1, except the boundary point, $\frac{\partial F}{\partial X} / \frac{\partial F}{\partial Y} < 1$.

At the boundary point (x^*, y^*), $\frac{\partial F}{\partial X} / \frac{\partial F}{\partial Y} = 1$.

Hence in R1: $\frac{\partial F}{\partial Y} > \frac{\partial F}{\partial X}$ and in R2: $\frac{\partial F}{\partial X} > \frac{\partial F}{\partial Y}$

At the boundary,

$$\frac{\partial F}{\partial Y} = \frac{\partial F}{\partial X}$$

.

$\frac{\partial F}{\partial X} = (2b^2 X)$

$\frac{\partial F}{\partial Y} = (2a^2 Y)$

Characterisation of movement from (x_p, y_p) to (x_{p+1}, y_{p+1}) in R1

- Let the current pixel be $(x_p,\ y_p)$; d_p= F(M_p),
  where M_p is the mid point of E and SE of (x_p, y_p)
  As $E = (x_p + 1, y_p)$ and $SE = (x_p + 1, y_p - 1)$,
  $M_p = (x_p + 1, y_p - 1/2)$;
- $d_p = \mathrm{F}(M_p) = \mathrm{F}(x_p + 1,\ Y_p - 1/2)$
  $= b^2(x_p + 1)^2 + a^2(y_p - 1/2)^2 - a^2b^2$
- **Case 1:** When E was chosen at iteration p ($d_p < 0$):
  $(x_{p+1}, y_{p+1}) = (x_p + 1, y_p)$
  $M_{p+1} = (x_p + 2, (y_p + y_p - 1)/2)$ (Mid point of E and SE of $(x_p + 1, y_p)$)
  $d_{p+1} = F(M_{p+1}) = \mathrm{F}(x_p + 2, y_p - 1/2)$
  $= b^2(x_p + 2)^2 + a^2(y_p - 1/2)^2 - a^2b^2$
  $d_{p+1} - d_p = b^2(2x_p + 3)$
  $d_{p+1} = d_p +\ b^2(2x_p + 3)$;

Let $(\Delta d)_{E1} = b^2(2x_p + 3)$
$d_{p+1} = d_p + (\Delta d)_{E1}$

- **Case 2:** When SE was chosen at iteration p ($d_p >= 0$):
  $(x_{p+1}, y_{p+1}) = (x_p + 1, y_p - 1)$
  $M_{p+1} = (x_p + 2, (y_p - 1 + y_p - 2)/2)$ (Mid point of E and SE of $(x_p + 1, y_p - 1)$)
  $d_{p+1} = F(M_{p+1}) = \mathrm{F}(x_p + 2, y_p - 3/2)$
  $= b^2(x_p + 2)^2 + a^2(y_p - 3/2)^2 - a^2b^2$
  $d_{p+1} - d_p = b^2(2x_p + 3) + a^2(-2y_p + 2)$
  $d_{p+1} = d_p + b^2(2x_p + 3) + a^2(-2y_p + 2)$;

  Let $(\Delta d)_{SE1} = b^2(2x_p + 3) + a^2(-2y_p + 2)$
  $d_{p+1} = d_p + (\Delta d)_{SE1}$

- Initial Condition:
  In R1, first point is (0, b)
  Let Initial value of d be $(d_{init)_{R1}} = F(M_1)$, Where M_1 is mid point of E and SE of (0,b)
  ie M_1 is mid point of $(1, b)$ and $(1, b-1)$
  $(d_{init})_{R1}$ = F(1, b - 1/2) $= b^2 + a^2(1/4 - b)$;
- Terminal condition: $\frac{\partial F}{\partial Y} = \frac{\partial F}{\partial X}$.
  ie Run the loop as long as $\frac{\partial F}{\partial Y} > \frac{\partial F}{\partial X}$

Problem with a fractional (floating point) value for $(d_{init})_{R1}$?
can be resolved by converting to closest integer

Characterisation of movement from (x_p, y_p) to (x_{p+1}, y_{p+1}) in R2

- Let the current pixel be $(x_p,\ y_p)$;
  d_p= F(M_p), where $M_p = ((x_p + 1 + x_p)/2, y_p - 1)$ (Mid point of SE and S of (x_p, y_p));
- $d_p = \mathrm{F}(M_p) = \mathrm{F}(x_p + 1/2, y_p - 1)$
  $= b^2(x_p + 1/2)^2 + a^2(y_p - 1)^2 - a^2 b^2$
  Let us find d_{p+1}
- **Case 1:** When S was chosen at iteration p ($d_p >= 0$):
  $(x_{p+1}, y_{p+1}) = (x_p, y_p - 1)$
  $M_{p+1} = (x_p + 1/2, y_p - 2)$ (Mid point of S and SE of $(x_p, y_p - 1)$)
  d_{p+1} =$F(M_{p+1})$=F$(x_p + 1/2, y_p - 2)$
  $=b^2(x_p + 1/2)^2 + a^2(y_p - 2)^2 - a^2 b^2$
  $d_{p+1} - d_p = a^2(-2y_p + 3)$
  d_{p+1}=d_p + $a^2(-2y_p + 3)$;

  Let $(\Delta d)_{S2}$= $a^2(-2Y_p + 3)$

$d_{p+1} = d_p + (\Delta d)_{S2}$

- **case 2:** When SE was chosen at iteration p ($d_p < 0$):
  $(x_{p+1}, y_{p+1}) = (x_p + 1, y_p - 1)$
  $M_{p+1} = ((x_p + 2 + x_p + 1)/2, y_p - 2)$ (Mid point of E and SE of $(x_p + 1, y_p - 1)$)
  $d_{p+1} = = F(M_{p+1}) = F(x_p + 3/2, y_p - 2)$
  $= b^2(x_p + 3/2)^2 + a^2(y_p - 2)^2 - a^2b^2$
  $d_{p+1} - d_p = b^2(2x_p + 2) + a^2(-2y_p + 3)$
  $d_{p+1} = d_p + b^2(2x_p + 2) + a^2(-2y_p + 3)$;
  Let $(\Delta d)_{SE2} = b^2(2x_p + 2) + a^2(-2y_p + 3)$

  $d_{p+1} = d_p + (\Delta d)_{SE1}$

- Initial Condition:
  Let the last point in R1 be (x_k, y_k), and hence the first point in R2 is $(x_{k+1}, y_{k+1}) = (x_k + 1/2, y_k - 1)$(Mid point of E and SE of (x_k, y_k))
  $(d_{init})_{R2} = F(x_k + 1/2, y_k - 1)$
  $= b^2(x_k + 1/2)^2 + a^2(y_k - 1)^2 - a^2b^2$;

- Termination Condition: $y_p = 0$, hence run the loop as long as $y_p > 0$

Problem with a fractional (floating point) value for $(d_{init})_{R2}$?
can be resolved by converting to closest integer

```
void MidPointEllipse(int a, int b, int value)
{

   //Intialization
   int  d2;
   int X = 0;
   int Y = b;
   sa = sqr(a);
   sb = sqr(b);
   int d1 = sb - sa*b + 0.25*sa;

   //Draw four points using symmetry
   EllipsePoints(X, Y, value); /* 4-way symmetrical pixel

   while ( sa*(Y - 0.5) > sb*(X + 1))/*Region R1 */
   {
     if (d1 < 0) /*Select E */
           d1 += sb*((X<<1) + 3);
```

RASTERIZATION OF ELLIPSE / ELLIPSE DRAWING (cont.)

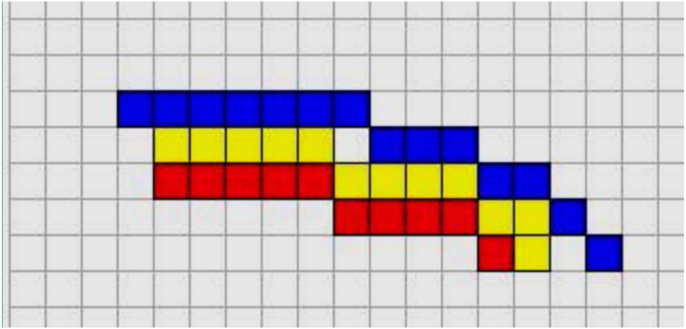
```
            else/*Select SE */
            {
              d1 += sb*((X<<1) + 3) + sa*(-(Y<<1) + 2);
              Y-- ;
            }
            X++ ;
            EllipsePoints(X, Y, value);
    }

    int d2 = sb*sqr(X + 0.5) + sa*sqr(Y - 1) - sa*sb;

    while ( Y > 0)    /*Region R2 */
    {
      if (d2 < 0)/*Select SE */
            {
              d2 += sb*((X<<1) + 2) +sa*(-(Y<<1) + 3);
              X++;
            }
```

RASTERIZATION OF ELLIPSE / ELLIPSE DRAWING (cont.)

```
            else /*Select S */
              d2 += sa*(-(Y<<1) + 3);
         Y-- ;
         EllipsePoints(X, Y, value);
         }
}


void EllipsePoints(int x, int y, int val)
{

    DrawPixel(x, y, val) //Q1
    DrawPixel(-x, -y, val) //Q3
    DrawPixel(-x, y, val) //Q2
    DrawPixel(x, -y, val) //Q4


}
```

RASTERIZATION OF ELLIPSE / ELLIPSE DRAWING (cont.)

In some cases the quality of the picture is not satisfactory



Possible Solution: 1) Increase resolution 2) Smooth the raster

Home Work

- Generalize the circle drawing algorithm for given center (a, b)
- Generalize the ellipse drawing algorithm for given center (a, b)
- Can you improve the midpoint ellipse drawing algorithm, avoiding multiplication inside the loop, without affecting the accuracy
- Trace the mid point line drawing algorithm for one line segment in each of the octants (The samples need to be plotted on graph sheet)
- Trace the mid point circle drawing algorithm for radius 5 and center $(1, 1)$(The samples need to be plotted on graph sheet)
- Trace mid point ellipse drawing algorithm for the center $(1, 1)$ and $a = 4$ and $b = 2$(The samples need to be plotted on graph sheet)

Rasterization of Region of Objects

Brute Force Technique when regular geometry or region bounded by $f(x, y) = 0$ is given

- Rectangle:
  - Given diagonal vertices of a triangle, find bounding box of the rectangle
  - For each point (x, y) inside the rectangle, DrawPixel(x,y)
- Triangle:
  - Given the vertices of a triangle, find bounding box of the triangle
  - check for each point (x, y) inside the box if it lies inside the triangle.
  - if yes, DrawPixel(x,y)

Boundary-fill algorithm

Flood-fill algorithm

Scan-line fill algorithm

Acknowledgements

- The slides have been adopted from NPTEL Lectures by Prof. Sukhendu Das. The due credits are acknowledged.

Thank You! :)