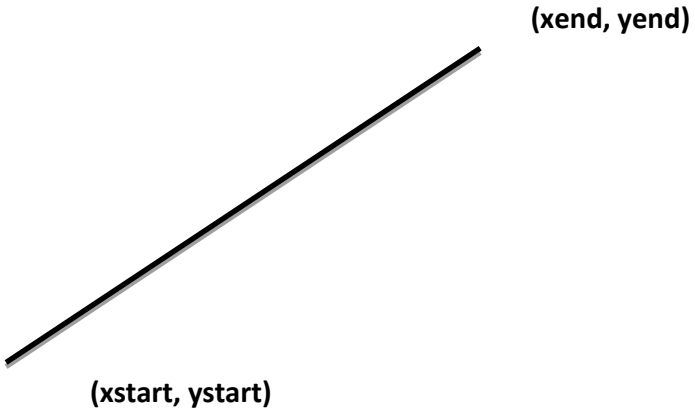
1.4 LINE DRAWING ALGORITHMS

Several line drawing algorithms are developed. Their basic objective is to enable visually satisfactory images in least possible time. This is achieved by reducing the calculations to a minimum. This is by using integer arithmetic rather than floating point arithmetic. This way of minimizing even a single arithmetic operation is important. This is because every drawing or image generated will have a large number of line segments in it and every line segment will have many pixels. So saving of one computation per pixel will save number of computations in generating an object. This in turn minimizes the time required to generate the whole image on the screen.

1.4.1DDA ALGORITHM (DIGITAL DIFFERENTIAL ANALYZER)

Suppose we are given the 2 end points of a line. (xstart, ystart) and (xend, yend).



(Fig: 1.34 - Line path between end point positions (xend, yend) and (xstart, ystart))

We know that the general equation of a line is y = mx + c. Where m is the slope of the line and c is the y- intercept. (x, y) is any point on the line.

We also know that the slope of the above line can be expressed as:-

$$m = \frac{yend - ystart}{xend - xstart}$$

Suppose we are given 2 points on the line as (xi, yi) and (xi+1, yi+1)then also slope,

$$m = \frac{yi+1 - yi}{xi+1 - xi}$$

We can say that

$$yi+1 - yi = dy$$

and

$$xi+1 - xi = dx$$

Then

$$m = \frac{dy}{dx}$$

That is dy = m dx

Thus for any given x interval dx along the line, we can compute the corresponding y interval dy.

Now suppose

xi+1 – xi = 1 or dx = 1

It means

xi+1 = xi + 1

Then

dy = m which means

yi+1 – yi= m

That is

yi+1 = yi + m → (1)

Thus a unit change in x changes y by m, which is constant for a line.

We know that if

xi+1 = xi + 1, then

yi+1 = yi +m

Initializing (xi, yi) with (xstart ,ystart), the line can be generated by incrementing the previous x values and solving the corresponding y value at each step, till xend is reached.at each step, we make incremental calculations based on the previous step. This is what is defined as incremental algorithm and often referred to as the Digital Differential Analyzer (DDA) algorithm.

We know

$$m = \frac{dy}{dx}$$

If |m| <= 1 which means |dy| <= |dx|, we sample the line at unit x intervals. But if |m| >1, in this a unit step in x creates a step in y that is greater than 1, which is not desirable. In this case we reverse the roles of x and y by sampling at unit y intervals as,

dy = yi+1 – yi = 1

implies yi+1 = yi + 1

$$m = \frac{dy}{dx}$$

implies

$$x_{i+1} - x_i = \frac{1}{m}$$

implies

$$x_{i+1} = x_i + \frac{1}{m} \quad \text{(2)}$$

$$y_{i+1} = y_i + m \quad \text{(3)}$$

These equations are based on the assumption that xstart<xend and ystart<yend. That is slope m is positive. But if it is not the case, we have to apply negative increment for the other possible cases.

The algorithm for rasterizing a line according to DDA logic is presented below.

DDA ALGORITHM

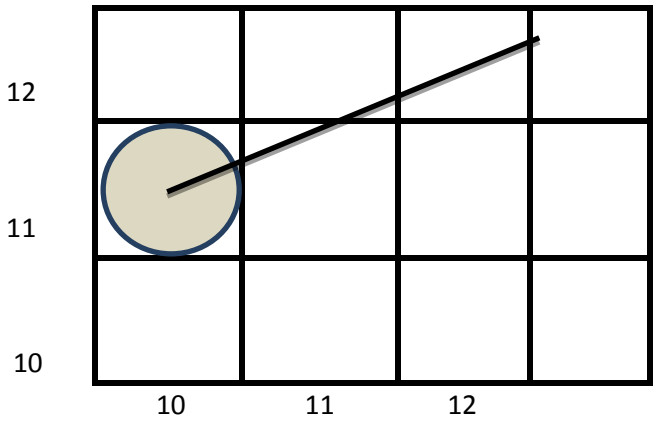
1. START
2. Get the values of the starting and ending co-ordinates i.e. ,(xa,ya) and (xb,yb).
3. Find the value of slope m
   m=dy/dx=((yb-ya)/(xb-xa))
4. If $|m| \leq 1$ then $\Delta x=1, \Delta y=m\Delta x$
   $x_{k+1}=x_k+1, y_{k+1}=y_k+m$
5. If $|m| \geq 1$ then $\Delta y=1, \Delta x=\Delta y/m$
   $x_{k+1}=x_k+1/m, y_{k+1}=y_k+1$
6. STOP

1.4.2 BRESENHAM'S LINE ALGORITHM

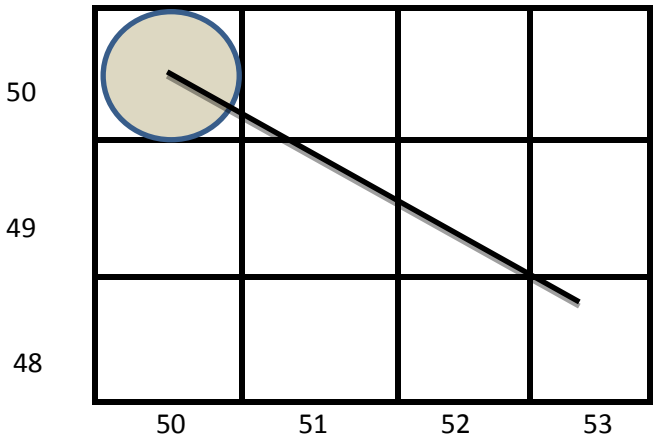
An accurate and efficient raster line-generating algorithm, developed by Bresenham, scans converts lines using only incremental integer calculations that can be adapted to display circles and other curves. Figures 1.35 and 1.36 illustrate sections of a display screen where straight line segments are to be drawn. The vertical axes show-scan-line positions, and the horizontal axes identify pixel columns. Sampling at unit ***x*** intervals in these examples, we need to decide which of two possible pixel positions is closer to the line path at each sample step. Starting from the left endpoint shown in Fig. 1.**35,** we need to determine at the next sample position whether to plot the pixel at position (11, 11) or the one at (11, 12). Similarly, Fig. 1.36 shows-a negative slope-line path starting from the left endpoint at pixel position (50, 50). In this one, do we select the next pixel position as (51,50) or as (51,49)? These questions are answered with Bresenham's line algorithm bytesting the sign of an integer parameter, whose value is proportional to the difference between the separations of the two pixel positions from the actual line path.

To illustrate Bresenharn's approach, we-first consider the scan-conversion process for lines with positive slope less than 1. Pixel positions along a line path are then determined by sampling at unit ***x*** intervals. Starting from the left endpoint (x_0, y_0)

of a given line, we step to each successive column ***(x*** position) and plot the pixel whose scan-line y value is closest to the line path. Assuming we have determined that the pixel at (x_k, y_k) is to be displayed, we next need to decide which pixel to plot in column ***x_{k+1},.***Our choices are the pixels at positions (X_k+l, y_k) and (x_{k+l}, y_k+l).



(Fig: 1.35 - Section of a display screen where a straight line segment is to be plotted, starting from the pixel at column 10 on scan line 11)



(Fig: 1.36 - Section of a display screen where a negative slope line segment is to be plotted, starting from the pixel at column 50 on scan line 50)

At sampling position x_k+l, we label vertical pixel separations from the mathematical line path as ***d_1***and ***d2*** (Fig. 1.**37).** Theycoordinates on the mathematical line at pixel column position x_k+l is calculated as

$$y=m(x_k+1)+b$$

Then

$$d_1=y-y_k$$
$$=m(x_k+1)+b-y_k$$

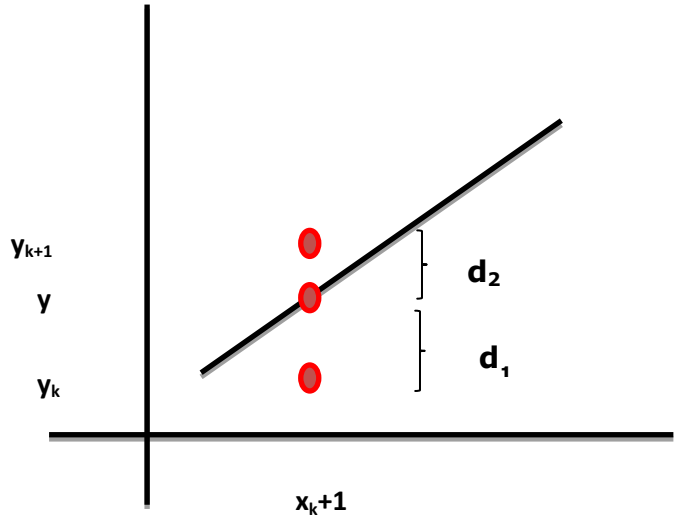
and

$d2=(y_k+1)-y$

$=y_k+1-m(x_k+1)-b$

The difference between these two separations is

$d1-d2=2m(x_k+1)-2y_k+2b-1$ ⟶ 4



(Fig: 1.37 - Distance between pixel positions and the line y coordinates at sampling position x_k+1)

A decision parameter ***pk***for the kth step in the line algorithm can be obtained by rearranging Eq. 4 so that it involves only integer calculations. We accomplish this by substituting **m** = Δy/Δx, where Δy and Δx are the vertical and horizontal separations of the endpoint positions, and defining:

$p_k=\Delta x(d_1-d_2)$

$=2\Delta y.x_k-2\Delta x.y_k+c$ ⟶ 5

The sign of p_k,is the same as the sign of d_l-d_2, since Δx>**0** for our example. Parameter cis constant and has the value 2Δy + Δx(2b - l), which is independentof pixel position and will be eliminated in the recursive calculations for p_k. If the pixel at $y_k$is closer to the line path than the pixel at y_k+1(that is, $d_l<d_2$), then decision parameter $p_k$is negative. In that case, we plot the lower pixel; otherwise, we plot the upper pixel.

Coordinate changes along the line occur in unit steps in either the ***x*** or ***y*** directions. Therefore, we can obtain the values of successive decision parameters using incremental integer calculations. At step k + 1, the decision parameter is evaluated from Eq. ***5***as

$p_{k+1}=2\Delta y.x_{k+1}-2\Delta x.y_{k+1}+c$

Subtracting **Eq.** 5 from the preceding equation, we have

$p_{k+1}-p_k=2\Delta y(x_{k+1}-x_k)-2\Delta x(y_{k+1}-y_k)$

But x_{k+1}, = x_k+ ***1,*** so that

$p_{k+1}=p_k+2\Delta y-2\Delta x(y_{k+1}-y_k)$

Where the term $y_{k+1} - y_k$ is either 0 or **1,** depending on the sign of parameter p_k.This recursive calculation of decision parameters is performed at each integer x position, starting at the left coordinate endpoint of the line. The first parameter, p_0, is evaluated from Eq. 5 at the starting pixel position (x_o, y_o) and with **m** evaluated as $\Delta y/\Delta x$:

We can summarize Bresenham line drawing for a line with a positive slope less than 1 in the following listed steps. The constants $2\Delta y$ and $2\Delta y - 2\Delta x$ are calculated once for each line to be scan converted, so the arithmetic involves only integer addition and subtraction of these two constants.

Bresenham's Line-Drawing Algorithm for I *m*I<1

1. Input the twoline endpoints and store the left endpoint in (x_o, y_o)

2. Load (x_O, y_O)into the frame buffer; that is, plot the first point.

3. Calculate constants Δx, Δy, $2\Delta y$, and $2\Delta y$ - 2Δx, and obtain the startingvalue for the decision parameter as

$p_o = 2\Delta y - \Delta x$

4. At each $x_k$along the line, starting at k = 0, perform the following test:

If $P_k < 0$, the next point to plot is $(x_k+1, y_k)$and

$P_{k+1} = P_k + 2\Delta y$

Otherwise, the next point to plot is $(x_k + 1, y_k + 1)$ and

$p_{k+1} = p_k + 2\Delta y - 2\Delta x$

5. Repeat step 4 Δ**x** times.

1.5 CIRCLE DRAWING

We know that the equation of a circle centered at the origin is $x^2 + y^2 = R^2$, where (0, 0) is the origin R is the radius and (x, y) is any point on the circle.

$x^2 + y^2 = R^2$

$y^2 = R^2 - x^2$

$y = + \sqrt{(R^2 - x^2)}$

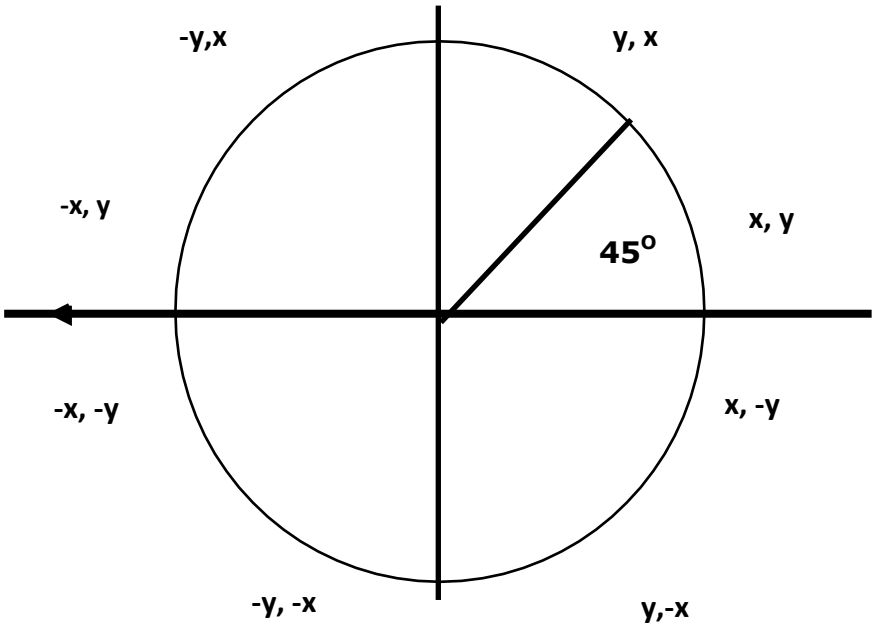
To draw a quarter circle, we can increment x from 0 to R in unit steps, solving for +y at each step.

This method will work, but it is inefficient because of the multiply and square root operations. Here the circle will have large gaps for values of x close to R, because the slope of the circle becomes infinite there.

Eight-way symmetry

We can improve the process of the previous section by taking greater advantage of the symmetry in a circle. Consider first a circle centered at the origin.

That is (0, 0). If the point (x, y) is on the circle the new can trivially compute seven other points on the circle as in **Fig: 1.38**



(Fig: 1.38, Eight symmetrical points on a circle)

We need to compute only one 45-degree segment to determine the circle completely. For a circle centered at the origin (0,0), the eight symmetrical points can be displayed with procedure circlepoints().

```
Void circlepoints (int x, int y)
{
        putpixel ( x, y);
        putpixel ( y, x);
        putpixel ( y, -x);
        putpixel ( x, -y);
        putpixel ( -x, -y);
        putpixel ( -y, -x);
        putpixel ( -y, x);
        putpixel ( -x, y);
}
```

This procedure can be easily generalized to the case of circles with arbitrary centers. Suppose the point (xcenter, ycenter) is the center of the circle. Then the above function can be modified as

```
Void circlepoints(xcenter, ycenter, x, y)
intxcenter, ycenter, x, y;
{
```

```
            putpixel ( xcenter + x, ycenter + y);
            putpixel ( xcenter + y, ycenter + x);
            putpixel ( xcenter + y, ycenter - x);
            putpixel ( xcenter + x, ycenter - y);
            putpixel ( xcenter - x, ycenter - y);
            putpixel ( xcenter - y, ycenter - x);
            putpixel ( xcenter -y, ycenter + x);
            putpixel ( xcenter - x, ycenter + y);
        }
```

1.5.1 DDA ALGORITHM

To write an algorithm to generate a circle of the form $(x-a)^2+(y-b)^2=r^2$ by the help of digital differential analyzer where (a,b) is the center of the circle and r is the radius.

1. START

2. Get the values of the center (a,b) and radius (r) of the circle.

3. Find the polar co-ordinates by

$x=a+r\cos\theta$

$y=b+r\sin\theta$

4. Plot the points(x,y) corresponding to the values of θ ,where θ lies between 0 and 360.

5.STOP

1.5.2 MIDPOINT CIRCLE ALGORITHM **(Bresenham's Circle Algorithm)**

As in the raster line algorithm, we sample at unit intervals and determine the closest pixel position to the specified circle path at each step. For a given radius ***r*** and screen center position ***(*x_c **,**y_c), we can first set up our algorithm to calculatepixel positions around a circle path centered at the coordinate origin ***(0,0)***.Then each calculated position **(x,** y) is moved to its proper screen position by adding $x_c$to **x** and $y_c$toy. Along the circle section from ***x* = *0*** to **x** = y in the first quadrant, the slope of the curve varies from ***0*** to -1. Therefore, we can take unit steps in the positive ***x*** direction over this octant and use a decision parameter to determine which of the two possible y positions is closer to the circle path at each step. Positions in the other seven octants are then obtained by symmetry.

To apply the midpoint method, we define a circle function:

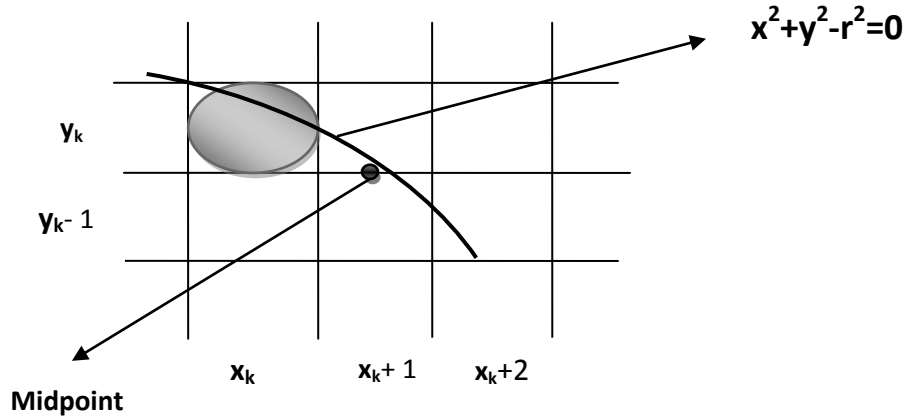
$$f_{circle}(x,y)=x^2+y^2-r^2$$

Any point ***(x,***y) on the boundary of the circle with radius r satisfies the equation $f_{circle}(x,y)= 0$. If the point is in the interior of the circle, the circle function is negative. And if the point is outside the circle, the circle function is positive. To summarize, the

relative position of any point ***(x. y)*** can be determined by checking the sign of the circle function:

$$f_{circle}(x,y) \begin{cases} <0 \text{ if } (x,y) \text{ is inside the circle boundary} \\ =0 \text{ if } (x,y) \text{ is on the circle boundary} \\ >0 \text{ if } (x,y) \text{ is outside the circle boundary} \end{cases}$$

The circle-function tests are performed for the midpositions between pixels near the circle path at each sampling step. Thus, the circle function is the decision parameter in the midpoint algorithm, and we can set up incremental calculations for this function as we did in the line algorithm.



(Fig: 1.39, Midpoint between candidate pixels at sampling position x_k+1 along a circular path)

Fig:1.39 shows the midpoint between the two candidate pixels at sampling position x_k+1. Assuming we have just plotted the pixel at (x_k, y_k), we next need to determine whether the pixel at position (x_k+1, y_k) or the one at position (x_k+1, y_k-1) *is* closer to the circle. Our decision parameter is the circle function 3-27 evaluated at the midpoint between these two pixels:

$$p_k = f_{circle}(x_k+1, y_k-(1/2))$$
$$= (x_k+1)^2+(y_k-(1/2))^2-r^2$$

If $p_k<0$, this midpoint is inside the circle and the pixel on scan line y_k is closer to the circle boundary. Otherwise, the midposition is outside or on the circle boundary, and we select the pixel on scan line y_k-1.

Successive decision parameters are obtained using incremental calculations. We obtain a recursive expression for the next decision parameter by evaluating the circle function at sampling position $x_{k+1}+1 = x_k+2$:

$$p_{k+1} = f_{circle}(x_{k+1}+1, y_{k+1}-(1/2))$$
$$= [(x_k+1)+1]^2+(y_{k+1}-(1/2))^2-r^2$$

Or

$$p_{k+1}=p_k+2(x_k+1)+(y^2_{k+1}-y^2_k)-(y_{k+1}-y_k)+1$$

Wherey_{k+1}, is either $y_k$or y_{k-1}, depending on the sign of p_k. increments for obtaining p_{k+1}, are either $2x_{k+1}+1$(if $p_k$is negative) or $2x_{k+1}+1-2y_{k+1}$. Evaluation of the terms $2x_{k+1}$,n d $2y_{k+1}$,can also *be* done incrementallyas

$$2x_{k+1}=2x_k+2$$

$$2y_{k+1}=2y_k-2$$

At the start position (0, r), these two terms have the values 0 and 2r, respectively. Each successive value is obtained by adding 2 to the previous value of 2xand subtracting 2 from the previous value of 2y.

The initial decision parameter is obtained by evaluating the circle function at the start position $(x_0, y_0) = (0, r)$:

$$p_0=f_{circle}(1,r-(1/2))$$
$$=1+(r-(1/2))^2-r^2$$

Or

$$p_0=(5/4)-r$$

If the radius r is **specified** as an integer, we can simply round $p_0$to

$p_0 = 1 - r$(for **r** an integer) , since all increments are integers.

Asin Bresenham's line algorithm, the midpoint method calculates pixel positions along the circumference of a circle using integer additions and subtractions, assuming that the circle parameters are specified in integer screen coordinate.

We can summarize the steps in the midpoint circle algorithm as follows.

Midpoint Circle Algorithm

1. Input radius r and circle center (x_c,y_c), and obtain the first point onthe circumference of a circle centered on the origin as

   $$(x_0,y_0)=(0,r)$$
2. Calculate the initial value of the decision parameter as

   $$p_0=(5/4)-r$$
3. At each x_k position, starting at $k = 0$, perform the following test: If $p_k<0$, the next point along the circle centered on **(0,0)** is (x_{k+1}, y_k) and

   $$p_{k+1}=p_k+2(x_{k+1})+1$$

   Otherwise, **the** next point along the circle is $(x_k + 1, y_k - 1)$and

   $$p_{k+1}=p_k+2(x_{k+1})+1-2(y_{k+1})$$

   Where$2x_{k+1}, = 2x_k + 2$and $2y_{k+1}, = 2y_k - 2$.
4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position (x, y) onto the circular path centered on(x_c,y_c) and plot the coordinate values:

   $$x=x+x_c, y=y+y_c$$
6. Repeat steps **3** through **5** until $x \geq y$.

Example:

Given a circle radius **r** = 10, we demonstrate the midpoint circle algorithm bydetermining positions along the circle octant in the first quadrant hum ***x*** = 0 to **x** = y. The initial value of the decision parameter is

$$p_0=1-r=-9$$

For the circle centered on the coordinate origin, the initial point is $(x_0, y_0) = (0, l0)$, and initial increment terms for calculating the decision parameters are:

$2x_0=0, 2y_0=20$

Successive decision parameter values and positions along the circle path are calculated using the midpoint method as

K	p_k	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
0	-9	(1,10)	2	20
1	-6	(2,10)	4	20
2	-1	(3,10)	6	20
3	6	(4,9)	8	18
4	-3	(5,9)	10	18
5	8	(6,8)	12	16
6	5	(7,7)	14	14

1.6 ELLIPSE-GENERATING ALGORITHMS

Loosely stated, an ellipse is an elongated circle. Therefore, elliptical curves can be generated by modifying circle-drawing procedures to take into account the different dimensions of an ellipse along the major and minor axes.

Properties of Ellipses

An ellipse is defined as the set of points such that the sum of the distances from two fixed positions (foci) is the same for all points (**Fig. 1.40***)*. If the distances to the two foci from any point P = *(x,* y) on the ellipse are labeled $d_l$and d_2, then the general equation of an ellipse can be stated as

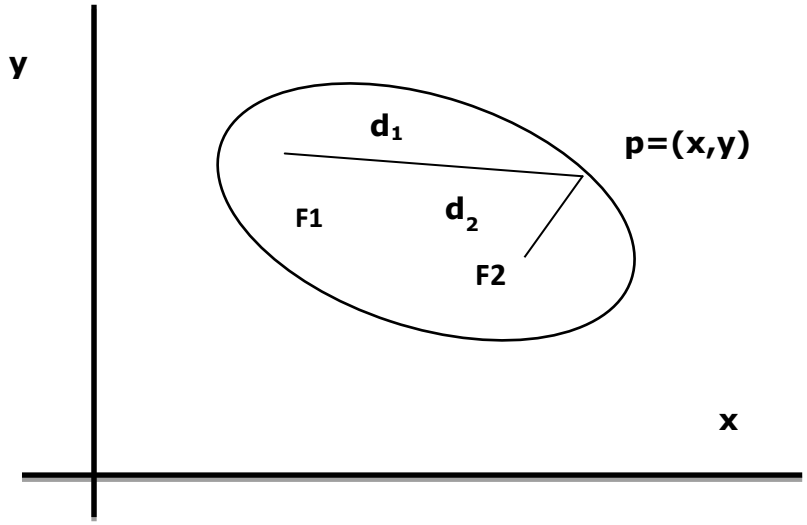
$$d_1 + d_2 = \text{constant}$$

Expressing distances $d_1$and $d_2$interms of the focal coordinates $F_1 = (x_1, y_1)$and $F_2 = (x_2, y_2)$, **we** have

$$\sqrt{((x-x_1)^2+(y-y_1)^2)} + \sqrt{((x-x_2)^2+(y-y_2)^2)} = \text{constant} \longrightarrow \text{A}$$

By squaring this equation, isolating the remaining radical, and then squaring again, we can rewrite the general ellipseequation in the form

$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0 \longrightarrow \text{B}$$



(Fig: 1.40, Ellipse generated about foci F1 and F2)

Where the coefficients ***A,*** B, C, ***D, E,*** and F are evaluated in terms of the focal coordinates and the dimensions of the major and minor axes of the ellipse. The major axis is the straight line segment extending from one side of the ellipse to the other through the foci. The minor axis spans the shorter dimension of the ellipse, bisecting the major axis at the halfway position (ellipse center) between the two foci.

An interactive method for specifying an ellipse in an arbitrary orientation is to input the two foci and a point on the ellipse boundary. With these three coordinate positions, we can evaluate the constant in **A**. Then, the coefficients in **B** can be evaluated and used to generate pixels along the elliptical path.

Ellipse equations are greatly simplified if the major and minor axes are oriented to align with the coordinate axes. In **Fig.1:41**, we show an ellipse in "standard position" with major and minor axes oriented parallel to the ***x*** and y axes. Parameter $\mathbf{r_x}$ for this example labels the semi major axis, and parameter $\mathbf{r_y}$ labels the semi minor axis. The equation of the ellipse shown in **Fig.1:42** can be written in terms of the ellipse center coordinates and parameters r_x and $\mathbf{r_y}$ as

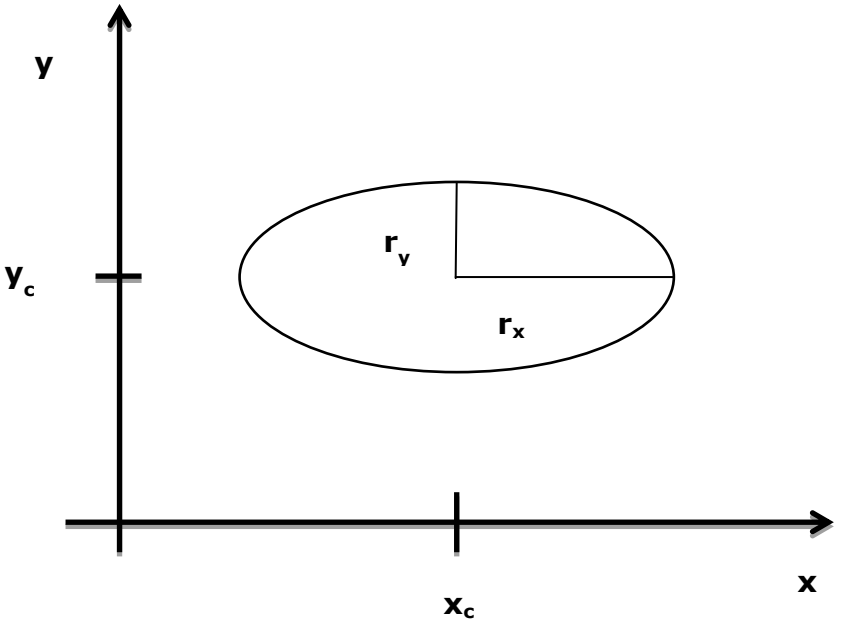
$$\mathbf{((x-x_c)/r_x)^2+((y-y_c)/r_y)^2=1} \longrightarrow \mathbf{C}$$

Using polar coordinates r and **θ,**we can also describe the ellipse in standard position with the parametric equations:

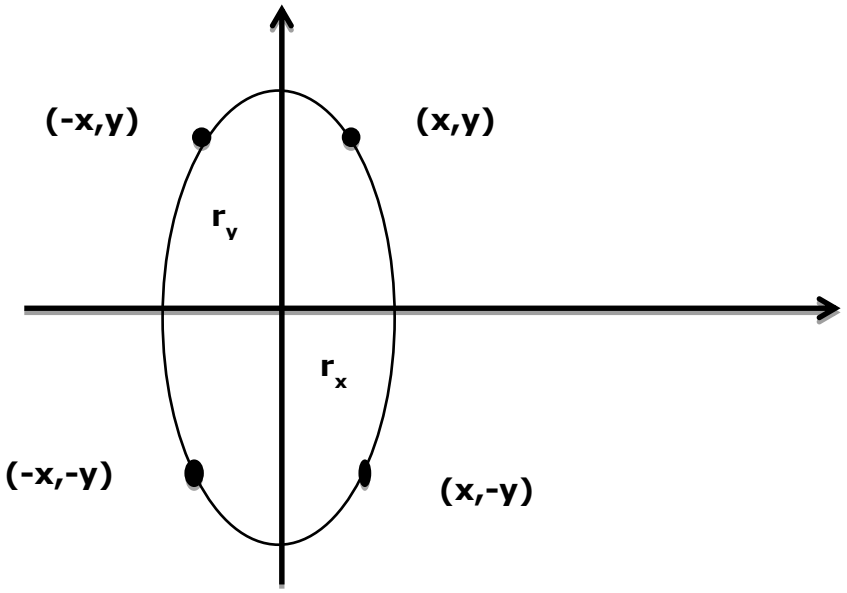
$$\mathbf{x}= x_c+r_x\cos\theta$$

$$\mathbf{y}= y_c+r_y\sin\theta$$

Symmetry considerations can be used to further reduce computations. An ellipse in standard position is symmetric between quadrants, but unlike a circle, it is not symmetric between the two octants of a quadrant. Thus, we must calculate pixel positions along the elliptical arc throughout one quadrant, and then we obtain positionsin the remaining three quadrants by symmetry **(Fig 1.42).**



(Fig: 1:41 Ellipse centered at(x_c,y_c)with semimajor axis $r_x$and semiminor axis r_y)



(Fig: 1:42Symmetry of an ellipse)

1.6.1DDA ALGORITHM

To write an algorithm to generate an ellipse using the Digital Differential AnalyzerAlgorithm (DDA).

Equation to the ellipse is

$((x-xc)/r_x)^2+((y-yc)/r_y)^2=1$

where (xc,yc) - center of the ellipse.

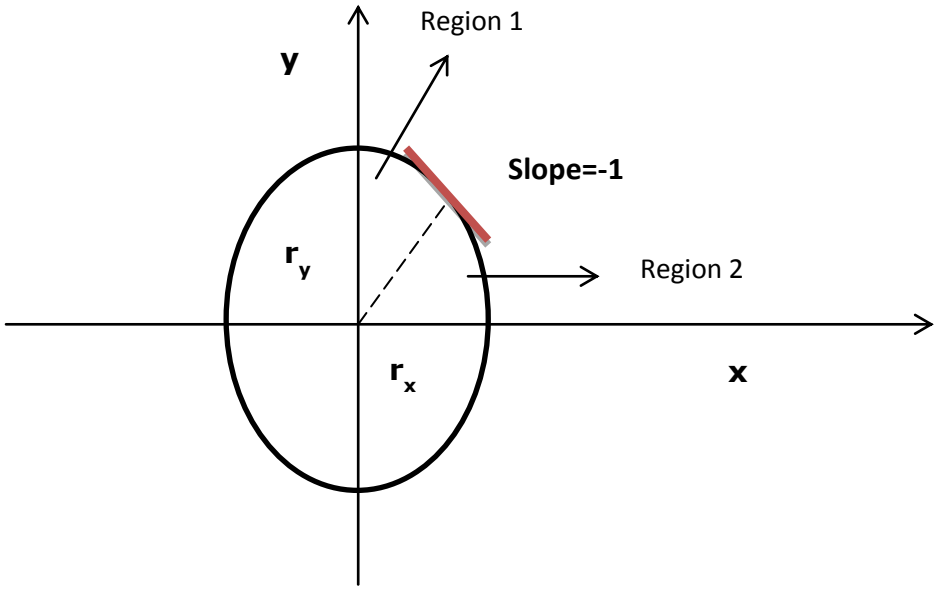
r_x- x radius of ellipse, r_y-y radius of ellipse.

1. START
2. Get the centre (xc,yc),x radius (rx) and y radius (ry) of the ellipse.
3. The polar co-ordinates on an ellipse are
   x=xc+rxcosθ
   y=yc+r y sinθ
4. Plot the point(x,y) corresponding to the values of θ where 0<=θ<=360
5. STOP

1.6.2 MIDPOINT ELLIPSE ALGORITHM (Bresenham's Circle Algorithm)

Our approach here is similar to that used in displaying araster circle. Given parameters r_x,$r_y$and (x_c,y_c), we determine points (x,y) for an ellipse in standard position centered on the origin, and then we shift the points so the ellipse is centered at (x_c,y_c). Ifwe wish also to display the ellipse in nonstandard position, we could then rotate the ellipse about its center coordinates to reorient the major and minor axes. For the present, we consider only the display of ellipses in standard position.

The midpoint ellipse method is applied throughout the first quadrant in two parts. **Fig 1:43** shows the division of the first quadrant according to the slope of an ellipse with $r_x < r_y$. We process this quadrant by taking unit steps in the **x**direction where the slope of the curve has a magnitude less than 1, and taking unit steps in they direction where the slop has a magnitude greater than 1.



(Fig: 1:43, Ellipse processing regions. Over region 1, the magnitude of the ellipse slope is less than 1; over region 2, the magnitude of the slope is greater than 1)

Regions 1 and 2 **(Fig. 1:43),**can be process in various ways. We can start at position $(0,r_y)$ and step clockwise along the elliptical path in the first quadrant, shifting from unit steps in ***x*** to unit steps in y when the slope becomes less than **-1.** Alternatively, we could start at $(r_x,0)$ and select points in a counterclockwise order,

shifting from unit steps in y to unit steps in **x** when the slope becomes greater than **-1.** With parallel processors, we could calculate pixel positions in the two regions simultaneously. As an example of a sequential implementation of the midpoint algorithm, we take the start position at ***(0,*** $\mathbf{r_y}$**)** and step along the ellipse path in clockwise order throughout the first quadrant.

We define an ellipse function from **C** with ***(*** x_c***,*** y_c***)*** = **(0,0)** as

$$f_{ellipse}(x,y)=r^2_y x^2+r^2_x y^2-r^2_x r^2_y$$

This has the following properties:

$f_{ellipse}(x,y)$ <0 if(x,y) is inside the ellipse boundary
=0 if(x,y) is on the ellipse boundary
>0if(x,y) is outside the ellipse boundary

Thus, the ellipse function $f_{ellipse}(x,y)$ serves as the decision parameter in the midpoint algorithm. At each sampling position, we select the next pixel along the ellipsepath according to the sign of the ellipse function evaluated at the midpoint between the two candidate pixels.

Midpoint Ellipse Algorithm

1. Input r_x, $r_y$and ellipse center ***(*** x_c***,*** y_c), and obtain the first point on an ellipse centered on the origin as

$$(x_0,y_0)= (0, r_y)$$

2. Calculate the initial value of thedecision parameter in region 1 as

$$p1_0=r^2_y-r^2_x r_y+(1/4)r^2_x$$

3. At each $x_k$position in region 1, starting at ***k*** = **0,** perform the following test:

If $pl_k< 0$, the next point along the ellipse centered on (0, 0) is**(**$\mathbf{x_{k+1}}$**,** y_k) and

$$p1_{k+1}=p1_k+2r^2_y x_{k+1}+r^2_y$$

Otherwise, the next point along the circle is ***(*** x_k+ 1, y_k- 1) and

$$p1_{k+1}=p1_k+2r^2_y x_{k+1}-2r^2_x y_{k+1}+r^2_y$$

with

$$2r^2_y x_{k+1}=2r^2_y x_k +2r^2_y$$

$$2r^2_x y_{k+1}=2r^2_x y_k -2r^2_x$$

and continue until $2r^2_y x \geq 2r^2_x y$.

4. Calculate the initial value of the decision parameter in region **2** using the last point ***(*** x_0***,*** y_0) calculated in region 1 as

$$p2_0=r^2_y(x_0+(1/2))^2+r^2_x(y_0-1)^2-r^2_x r^2_y$$

5. At each $y_k$position in region **2,** starting at k = 0, perform the followingtest: If $p2_k$>0, the next point along the ellipse centered on (0, 0) is***(*** x_k***,*** y_k-1) and

$$p2_{k+1}=p2_k-2r^2_x y_{k+1}+r^2_x$$

Otherwise, the next point along the circle *is* ***(*** x_k+ **1,** y_k - **1)** and

$$p2_{k+1}=p2_k+2r^2_y x_{k+1}-2r^2_x y_{k+1}+r^2_x$$

Using the same incremental calculations for **x** and y as in region 1.

6. Determine symmetry points in the other three quadrants.

7. Move each calculated pixel position ***(x,*** y) onto theelliptical path centered on***(x_c,*** y_c) and plot the coordinate values:

$x=x+x_c$ $y=y+y_c$

8. Repeat the steps for region 1 until $2r^2_y x \geq 2r^2_x y$.

Example:

Given input ellipse parameters r_x= 8 and r_y= 6, we illustrate the steps in themidpoint ellipse algorithm by determining raster positions along the ellipse path in the first quadrant. Initial values and increments for the decision parameter calculations are

$2r^2_y x$= **0** (with increment $2r^2_y$= ***72)***

$2r^2_x y=2r^2_x r_y$**(**withincrement-$2r^2_x$**=-128)**

For region 1: The initial point for the ellipse centered on the origin is **(x_0,** y_0) = **(0,6),**and the initial decision parameter value is

$p1_0=r^2_y-r^2_x r_y+(1/4)r^2_x$= ***-332***

Successive decision parameter values and positions along the ellipse path are calculated using the midpoint method as

k	**$p1_k$**	**(x_{k+1},y_{k+1})**	**$2r^2_y x_{k+1}$**	**$2r^2_x y_{k+1}$**
0	-332	(1,6)	72	768
1	-224	(2,6)	144	768
2	-44	(3,6)	216	768
3	208	(4,5)	288	640
4	-108	(5,5)	360	640
5	288	(6,4)	432	512
6	244	(7,3)	504	384

We now **move** out of region 1, since $2r^2_y x>2r^2_x y$

For region 2, the initial point is **(x_0, *y_0)** = **(7,3)** and the initial decision parameter is

$p2_0=f(7+(1/2),2)=-151$

The remaining positions along the ellipse path in the first quadrant are then calculated as

k	**$P2_k$**	**(x_{k+1},y_{k+1})**	**$2r^2_y x_{k+1}$**	**$2r^2_x y_{k+1}$**
0	-151	(8,2)	576	256
1	233	(8,1)	576	128
2	745	(8,0)	-	-

1.7 PARALLEL LINE ALGORITHMS

With a parallel computer, we can calculate pixel positions along a line path simultaneously by partitioning the computations among the various processors available. One approach to the partitioning problem is to adapt an existing sequential algorithm to take advantage of multiple processors. Alternatively, we can look for other ways to set up the processing so that pixel positions can be calculated efficiently in parallel. An important consideration in devising a parallel algorithm is to balance the processing load among the available processors.

Given $\mathbf{n_p}$ processors, we can set up a parallel Bresenham line algorithm by subdividing the line path into $\mathbf{n_p}$ partitions and simultaneously generating line segments in each of the subintervals. For a line with slope $\mathbf{0} < \boldsymbol{m} < 1$ and left endpoint coordinate position ***(x0,*** y0), we partition the line along the positive ***x*** direction. The distance between beginning **x** positions of adjacent partitions can be calculated as

$$\Delta x_p=(\Delta x+n_p-1)/n_p$$

where Δx is the width of the line, and the value for partition width $\Delta \boldsymbol{x_p}$ is computed using integer division. Numbering the partitions, and the processors, as **0,1,2,** up to n, - 1, we calculate the starting **x** coordinate for the kth partition as

$$x_k=x0+k\Delta x_p$$

As an example, suppose $\Delta x=15$ and we have $n_p=4$ processors. Then the width of the partitions is 4 and the starting ***x*** values for the partitions are x0, x0 + **4,** ***x0*** +8, and x0 + 12. With this partitioning scheme, the width of the last (rightmost) subinterval will be smaller than the others in some cases. In addition, if the line endpoints are not integers, truncation errors can result in variable width partitions along the length of the line.

To apply Bresenham's algorithm over the partitions, we **need** the initial value for the y coordinate and the initial value for the decision parameter in each partition. The change Δy_p, in the y direction over each partition is calculated from the line slope **m** and partition width Δx_p:

$$\Delta y_p= m\Delta x_p$$

At the kth partition, the starting y coordinate is then

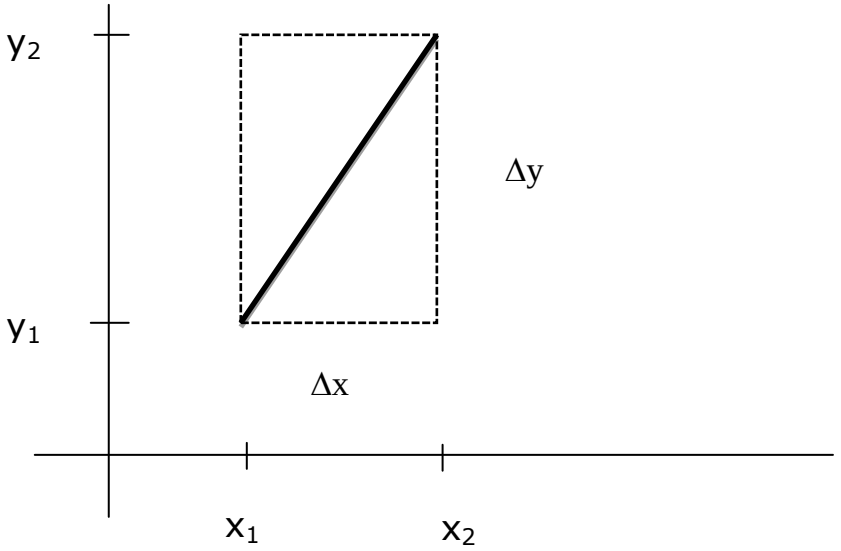
$$y_k=y0+\text{round}(k\Delta y_p)$$

The initial decision parameter for Bresenham's algorithm at the start of the **kth** subinterval is obtained from

$$p_k=(k\Delta x_p)(2\Delta y)-\text{round}(k\Delta y_p)(2\Delta x)+2\Delta y-\Delta x$$

Each processor then calculates pixel positions over its assigned subinterval using the starting decision parameter value for that subinterval and the starting coordinates (x0,y0).We can also reduce the floating-point calculations to integer

arithmetic in the computations for starting values y_k and $\mathbf{p_k}$ by substituting $\boldsymbol{m}$ =Δy/Δx and rearranging terms. The extension of the parallel Bresenham algorithm to a line with slope greater than 1 is achieved by partitioning the line in the y direction and calculating beginning $\boldsymbol{x}$ values for the partitions. For negative slopes, we increment coordinate values in one direction and decrement in the other.



(Fig: 1:44 Bounding box for a line with coordinate extents Δx and Δy)