Page no
Date: / /

SCAN- CONVERSION

Many pictures, from 2D drawings to projected views of 3D objects, consist of graphical primitives such as points, lines, circles and filled polygons. These picture components are often defined in a continuous space at a higher level of abstraction than individual pixels in the discrete image space. For instance, a line is defined by its two endpoints and the line equation, whereas a circle is defined by its radius, center position, and the circle equation. It is the responsibility of the graphics system or the application program to convert each primitive from its geometric definition into a set of pixels that make up the primitive in the image space. This conversion task is generally referred to as scan conversion or rasterization.

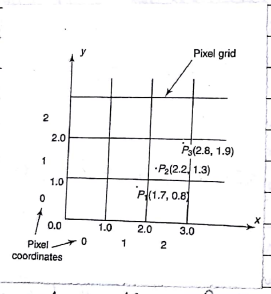
Rasterization is the task of taking an image described in a vector graphics format (shapes) and converting it into a raster image (pixels or dots) for output on a video display or printer, or for storage in a bitmap file format.

Scan-Converting a Point

A mathematical point (x, y) where x and y are real numbers within image area, needs to be scan-converted to a pixel at location (x', y'). This may be done by making x' to be the integer part of x, and y' the integer part of y.

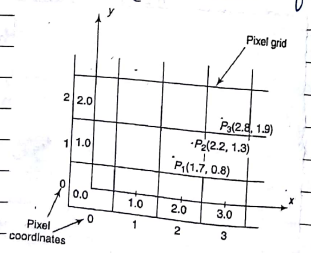
for example, $x' = \text{Floor}(x)$ and $y' = \text{Floor}(y)$, where function Floor returns the largest integer that is less

than or equal to the argument. Doing so in essence places the origin of a continuous coordinate system for (x, y) at the lower left corner of the pixel grid in the image space. (See the following figure). All points that satisfy $x' \leq x < x'+1$ and $y' \leq y < y'+1$ are mapped to pixel (x', y'). For example, point $P_1(1.7, 0.8)$ is represented by pixel (1,0). Points $P_2(2.2, 1.3)$ and $P_3(2.8, 1.9)$ are both represented by pixel (2,1).



Scan-Converting Points

Another approach is to align the integer values in the coordinate system for (x, y) with the pixel coordinates. (See the following figure).

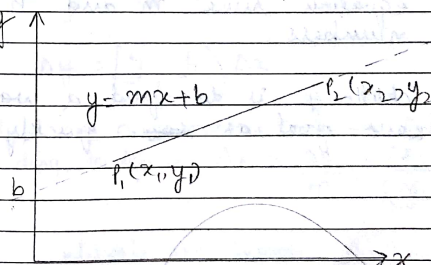


Scan-Converting Points

Here we scan convert (x,y) by making $x' = \text{Floor}(x + 0.5)$ and $y' = \text{Floor}(y + 0.5)$. This essentially places the origin of the coordinate system for (x,y) at the center of pixel (0,0). All points that satisfy $x' - 0.5 \leq x < x' + 0.5$ and $y' - 0.5 \leq y < y' + 0.5$ are mapped to pixel (x', y'). This means that points P_1 and P_2 are now both represented by pixel (2,1), whereas point P_3 is represented by pixel (3,2).

Scan-Converting a Line

A line in computer graphics typically refers to a line segment, which is a portion of a straight line that extends indefinitely in opposite directions. It is defined by its two endpoints and the line equation $y = mx + b$, where m is called the slope and b the y-intercept of the line. In the following figure the two endpoints are described by $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$. The line equation describes the coordinates of all the points that lie between the two endpoints.



Defining a line

Direct use of Line Equation

A simple approach to scan-converting a line is to first scan-convert P_1 and P_2 to pixel-coordinates, (x_1', y_1') and (x_2', y_2') respectively;

Then set $m = \frac{(y_2' - y_1')}{(x_2' - x_1')}$

& $b = y_1' - m x_1'$ if $|m| \leq 1$.

Then for every integer value of x between and excluding x_1', x_2' calculate the corresponding value of y using the equation and scan-convert (x, y).

If $|m| > 1$, then for every integer value of y between and excluding y_1' and y_2', calculate the corresponding value of x using the equation and scan-convert (x, y).

While this approach is mathematically sound it involves floating-point computation (multiplication and addition) in every step that uses the line equation since m and b are generally real numbers.

The challenge is to find a way to achieve the same goal as ~~easy~~ quickly as possible.

DIGITAL DIFFERENTIAL ANALYZER (DDA) ALGORITHM

→ The DDA algorithm is an incremental algorithm (scan-conversion method). Such an approach is characterized by performing calculations at each step using results from the preceding step.

→ The Digital Differential Analyzer name comes from the fact that we use the same technique as a numerical method for solving differential equations.

Slope of the straight line is given as

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

This differential equation can be used to obtain a rasterized straight line.

For any given Δx (x interval) along a line, we can compute the Δy (y interval) as

$$\Delta y = m \, \Delta x$$

ie. $$\Delta y = \left(\frac{y_2 - y_1}{x_2 - x_1}\right) \Delta x$$

Similarly, we can obtain the x interval Δx corresponding to Δy as $\Delta x = \frac{\Delta y}{m} = \left(\frac{x_2 - x_1}{y_2 - y_1}\right) \Delta y$

Once, the intervals are known, the next values for x and y are obtained as

$$x_{i+1} = x_i + \Delta x$$

$$x_{i+1} = x_i + \Delta x$$

$$x_{i+1} = x_i + \left(\frac{x_2 - x_1}{y_2 - y_1}\right)\Delta y$$

and $$y_{i+1} = y_i + \Delta y$$

$$y_{i+1} = y_i + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)\Delta x$$

for simple DDA either Δx or Δy, whichever is larger is chosen as some raster unit. i.e.

If $|\Delta x| \geq |\Delta y|$

then $\Delta x = 1$

else $\Delta y = 1$

If $\Delta x = 1$ then

$$y_{i+1} = y_i + \frac{y_2 - y_1}{x_2 - x_1}$$

and $x_{i+1} = x_i + 1$

If $\Delta y = 1$

then $y_{i+1} = y_i + 1$

$$x_{i+1} = x_i + \frac{x_2 - x_1}{y_2 - y_1}$$

DDA Algorithm (Stepwise).

1. Read the line endpoints (x_1, y_1) and (x_2, y_2)
2. $\Delta x = |x_2 - x_1|$
   $\Delta y = |y_2 - y_1|$
3. If $(\Delta x > \Delta y)$
   length $= \Delta x$
   else
   length $= \Delta y$
4. Select the raster unit, i.e.,
   $$m = \frac{(x_2 - x_1)}{length}$$
   $$n = \frac{(y_2 - y_1)}{length}$$
5. $x = x_1 + 0.5 * sign(m)$
   $y = y_1 + 0.5 * sign(n)$

Sign function makes the algorithm work in all quadrant. It returns $-1, 0, 1$ depending on whether its agreement is $<0, =0, >0$ respectively. The factor 0.5 makes it possible to round the values in the integer function rather than truncating them.

6. Now plot the points
   i = 1
   while (i ≤ length)
   {
   Plot (integer (x), integer (y))
   x = x + m
   y = y + n
   i = i + 1
   }
7. Stop

Page no
Date: / /

Advantages of DDA Algorithm

1. DDA Algorithm is faster than the direct use of the line equations since it calculates points on the line without any floating point multiplication.

2. It is a simplest algorithm and does not require special skills for implementation.

Disadvantages of DDA Algorithm

1. It is orientation dependent, due to this the end point accuracy is poor.

2. A floating-point addition is still needed in determining each successive point which is time consuming. Furthermore, cumulative error due to limited precision in the floating point representation may cause calculated points to drift away from their true position when the line is relatively long.

Page no
Date: / /

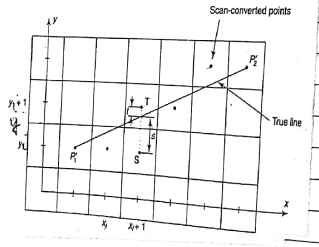
Bresenham's Line Algorithm

An accurate and efficient raster line algorithm, developed by Bresenham, scan converts lines using only incremental integer calculations that can be adapted to display circles and other curves.

Bresenham's line algorithm is a highly efficient incremental method for scan-converting lines. It produces mathematically accurate results using only integer addition, subtraction and multiplication by 2, which can be accomplished by a simple arithmetic shift operation.

The method works as follows :-

Assume that we want to scan-convert the line shown in the following figure, where $0 < m < 1$.



Scan-Converting a line using Bresenham Algorithm

Page no
Date: / /

We start with pixel $P_1'(x_1', y_1')$, then select subsequent pixels as we work our way to the right, one pixel position at a time in the horizontal direction towards $P_2'(x_2', y_2')$.

Once a pixel is chosen at any step, the next pixel is either the one to its right (which constitutes a lower bound for the line) or the one to its right and up (which constitutes an upper bound for the line) due to the limit on m. The line is best approximated by those pixels that fall the least distance from its true path between P_1' and P_2'.

Using the notation of figure 1, the coordinate of the last chosen pixel upon entering step i are (x_i, y_i).

Our task is to choose the next one between the bottom pixel S and the top pixel T.
If S is chosen, we have $x_{i+1} = x_i + 1$ and $y_{i+1} = y_i$

If T is chosen, we have

$$x_{i+1} = x_i + 1$$

& $$y_{i+1} = y_i + 1$$

The actual y coordinate of the line at $x = x_{i+1}$ is $y = mx_{i+1} + b$

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

The distance from S to the actual line in the y direction is

$$s = y - y_i$$

The distance from T to the actual line in the y direction is

$$t = (y_i + 1) - y$$

Now, consider the difference between these two distance values :-

$$s - t$$

when $(s-t) < 0$, we have $s < t$ and the closest pixel is S.

Conversely, when $(s-t) > 0$, we have $s > t$ and the closest pixel is T.

We also choose T when $(s-t) = 0$

This difference is

$$(s-t) = (y - y_i) - [(y_i + 1) - y]$$
$$= y - y_i - y_i - 1 + y$$
$$= 2y - 2y_i - 1$$

Putting the value of y $\{y = m(x_i + 1) + b\}$, we get,

$$(s-t) = 2m(x_i + 1) + 2b - 2y_i - 1$$

Substituting m by $\frac{\Delta y}{\Delta x}$ and introducing a decision variable $d_i = \Delta x (s-t)$.

$$(s-t) = \frac{2\Delta y}{\Delta x}(x_i + 1) + 2b - 2y_i - 1$$

Multiplying both sides by Δx we get,

$$\Delta x (s-t) = \Delta x \left[\frac{2\Delta y}{\Delta x}(x_i + 1) + 2b - 2y_i - 1\right]$$

$$\Delta x (s-t) = 2\Delta y (x_i + 1) + 2b - 2y_i - 1$$

$$\Delta x (s-t) = 2\Delta y x_i + 2\Delta y + 2b\Delta x - 2y_i\Delta x - \Delta x$$

$$\Delta x (s-t) = 2\Delta y x_i - 2y_i\Delta x + 2\Delta y + 2b\Delta x - \Delta x$$

$$\Delta x (s-t) = 2\Delta y x_i - 2y_i\Delta x + C$$

$$d_i = 2\Delta y x_i - 2y_i + C$$

where d_i is the decision variable as mentioned above, $d_i = \Delta x (s-t)$

& constant $C = 2\Delta y + \Delta x(2b - 1)$

Similarly, we can write the decision variable d_{i+1} for the next step as

$$d_{i+1} = 2\Delta y x_{i+1} - 2\Delta x y_{i+1} + C$$

Then,

$$d_{i+1} - d_i = 2\Delta y\, x_{i+1} - 2\Delta x\, y_{i+1} + c - (2\Delta y\, x_i - 2\Delta x\, y_i + c)$$

$$= 2\Delta y\, x_{i+1} - 2\Delta x\, y_{i+1} + c - 2\Delta y\, x_i + 2\Delta x\, y_i - c$$

$$d_{i+1} - d_i = 2\Delta y\,(x_{i+1} - x_i) - 2\Delta x\,(y_{i+1} - y_i)$$

since, $x_{i+1} = x_i + 1$, we have,

$$d_{i+1} = d_i + 2\Delta y\,(x_i + 1 - x_i) - 2\Delta x\,(y_{i+1} - y_i)$$

$$d_{i+1} = d_i + 2\Delta y - 2\Delta x\,(y_{i+1} - y_i)$$

If the chosen pixel is the Top pixel T (meaning that $d_i \geq 0$)

then, $y_{i+1} = y_i + 1$

& so, $d_{i+1} = d_i + 2\Delta y - 2\Delta x$.

If the chosen pixel is the bottom pixel S (meaning $d_i < 0$),

then $y_{i+1} = y_i$

& so, $d_{i+1} = d_i + 2\Delta y$

Hence, we have,

$$d_{i+1} = \begin{cases} d_i + 2(\Delta y - \Delta x) & \text{if } d_i \geq 0 \\ d_i + 2\Delta y & \text{if } d_i < 0 \end{cases}$$

finally, we calculate d_1, the base case value for this recursive formula, from the original definition of the decision variable d_i

$$d_i = 2\Delta y\, x_i - 2y_i + 2\Delta y + 2b - 1$$

$$= 2\Delta y\,(x_i + 1) - 2y_i + 2\Delta y + 2b - 1$$

$$= \Delta x\left[2\frac{\Delta y}{\Delta x}(x_i + 1) - 2y_i + 2\Delta x \cdot \frac{\Delta y}{\Delta x} + 2b - 1\right]$$

$$= \Delta x\,[2m x_i + 2m - 2y_i + 2m\Delta x + 2b - 1]$$

$$d_1 = \Delta x\,[2[m x_1 + m - y_1 + m\Delta x + b] - 1]$$

$$= \Delta x\,[2(m + m\Delta x) - 1]$$

$$= \Delta x\,[2m + 2m\Delta x - 1]$$

$$= 2\Delta y +$$

P. T. O.

$$d_i = \Delta x\left[2m(x_i+1) + 2b - 2y_i + 1\right]$$

Putting $i=1$ $\quad d_1 = \Delta x\left[2m(x_1+1) + 2b - 2y_1 - 1\right]$

$$d_1 = \Delta x\left[2(mx_1 + m + b - y_1) - 1\right]$$

As, $y_i = mx_i + b$

So, $y_1 = mx_1 + b$

$$mx_1 + b - y_1 = 0$$

Putting this value in the above d_1 expression.

$$d_1 = \Delta x\left[2m - 1\right]$$

$$= 2m.\Delta x - \Delta x$$

$$= 2\frac{\Delta y}{\Delta x}.\Delta x - \Delta x \quad \{As\ m = \frac{\Delta y}{\Delta x}\}$$

$$d_1 = 2\Delta y - \Delta x \quad \{Base\ case\ value\}$$

Bresenham's Algorithm for scan-converting a line from $P_1'(x_1', y_1')$ to $P_2'(x_2', y_2')$ with $x_1' < x_2'$ and $0 < m < 1$ is as follows:

```
int x = x1', y = y1';
int dx = x2' - x1';
int dy = y2' - y1';
int dT = 2(dy - dx);
int dS = 2dy;
int d = 2dy - dx;
setPixel(x,y);
while (x < x2') {
    x++;
    if (d<0)
        d = d + dS;
    else {
        y++;
        d = d + dT;
    }
    setPixel(x, y);
}
```

Here, we first initialize decision variable d and set Pixel P_1'. During each iteration of the while loop, we increment x to the next horizontal position, then use the current value

of d to select the bottom or top (increment) pixel and update d, and at the end set the chosen pixel.

As for the lines that have other m values we can make use of the fact that they can be mirrored either horizontally, vertically, or diagonally into this 0° to 45° angle range.

for example, a line from (x_1', y_1') to (x_2', y_2') with $-1 < m < 0$ has a horizontally mirrored counterpart, from $(x_1', -y_1')$ to $(x_2', -y_2')$ with $0 < m < 1$. We can simply use the algorithm to scan-convert this counterpart, but negate the y coordinate at the end of each iteration to set the right pixel for the line.

for a line whose slope is in the 45° to 90° range, we can obtain its mirrored counterpart by exchanging the x and y coordinates of its endpoints. We can then scan-convert this counterpart but we must exchange x & y in the call to setPixel.

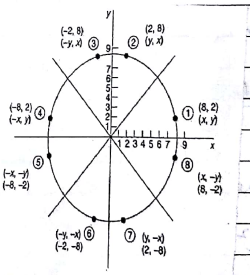
Scan-Converting a Circle

→ A circle is a symmetrical figure.

→ Any circle generating algorithm can take advantage of the circle's symmetry to plot eight points for each value that the algorithm calculates.

→ Eight-way symmetry is used by reflecting each calculated point around each 45° axis.

⇒ For example, if point 1 in the following figure were calculated with a circle algorithm, seven more points could be found by reflection. The reflection is accomplished by reversing the x, y coordinates as in point 2, reversing the x, y coordinates and reflecting about the y-axis as in point 3, reflecting about the y-axis as in point 4, switching the signs of x & y as in point 5, reversing the x, y coordinates, reflecting about the y-axis and reflecting about the x-axis as in point 6, reversing the x, y coordinates and reflecting about the y axis as in point 7, and reflecting about the x-axis as in point 8.



Eight-way symmetry of a Circle

Date: / /

To summarize

$P_1 = (x, y)$
$P_2 = (y, x)$
$P_3 = (-y, x)$
$P_4 = (-x, y)$
$P_5 = (-x, -y)$
$P_6 = (y, -x)$
$P_7 = (y, -x)$
$P_8 = (x, -y)$

Defining a Circle :-

There are 2 (two) standard methods of mathematically defining a circle centered at origin. The first method defines a circle with the second-order polynomial equation.

Representation of Circles

1) Polynomial Method
2) Trigonometric Method

Polynomial Method

In this method, circle is represented by a polynomial equation

$$x^2 + y^2 = r^2$$

where x is the x coordinate
y is the y coordinate
r is the radius of circle.

Page no
Date: / /

Polynomial equation can be used to find y-coordinate for the known x-coordinate.

∴ The scan converting circle using polynomial method is achieved by stepping x from 0 to $r/\sqrt{2}$ and each y coordinate is found by evaluating $\sqrt{r^2 - x^2}$ for each step of x.

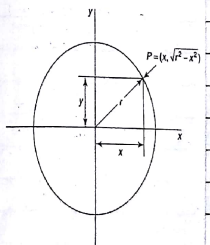
This generates the 1/8 portion of the circle. Remaining part of the circle can be generated just be reflection.

Trigonometric Method.

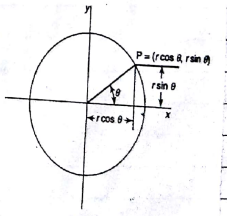
In this method, the circle is represented by use of trigonometric functions

$$x = r\cos\theta$$
$$y = r\sin\theta$$

where x = x coordinate, θ = current angle
y = y coordinate, r = radius of the circle



Circle defined with a second degree Polynomial equation



Circle defined with Trigonometric Functions

Page no
Date: / /

The scan converting circle using trigonometric method is achieved by stepping θ from 0 to $\pi/4$ radians, and each value of x and y is calculated.

This method is more inefficient than the polynomial method because the computation of the values of $\sin\theta$ and $\cos\theta$ is even more time consuming than the calculations required in the polynomial method.

Page no
Date: / /

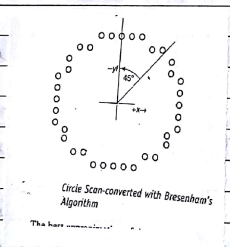
Bresenham's Circle Algorithm.

→ If a circle is to be plotted efficiently, the use of trigonometric and power functions must be avoided.

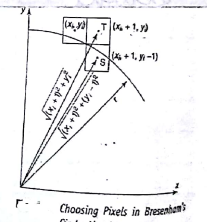
→ And, as with the generation of a straight line, it is also desirable to perform the calculations necessary to find the scan converted points with only integer addition, subtraction and multiplication by powers of 2.

→ Bresenham's circle algorithm allows these goals to be met.

Scan-Converting a Circle using Bresenham's algorithm works as follows :-



Circle Scan-converted with Bresenham's Algorithm



Choosing Pixels in Bresenham's Circle Algorithm

If the eight-way symmetry of a circle is used to generate a circle, points will be only have to be generated through a 45° angle.

And, if points are generated from 90° to 45°, moves will be made only in the +x and −y direction. (see figure 1).

The best approximation of true circle will be described by those pixels in the raster that fall the least distance from the true circle. See figure 2, notice that if points are generated from 90° and 45°, each new point closest to the true circle can be found by taking either of two actions:-

1) Move in the x-direction one unit or

2) move in the x-direction one unit and move in the negative y-direction one unit.

Therefore, a method of selecting between these two choices is all that is necessary to find the points closest to the true circle.

Assume that (x_i, y_i) are the coordinates of the last scan-converted pixel upon entering step 2

Let the distance from the origin to pixel S squared minus the distance to the true circle squared = D(T)

Then, let the distance from the origin to pixel S squared minus the distance to the true circle squared = D(S)

As, the coordinates of T are (x_i+1, y_i) and those of S are (x_i+1, y_i-1), the following expressions can be developed.

$$D(T) = (x_i+1)^2 + y_i^2 - r^2$$

$$D(S) = (x_i+1)^2 + (y_i-1)^2 - r^2$$

This function D provides a relative measurement of the distance from the center of a pixel to the true circle.

Since, D(T) will always be positive (∵ T is outside the true circle),

& D(S) will always be negative (∵ S is inside the true circle). So, a decision variable d_i may be defined as follows:-

$$d_i = D(T) + D(S)$$

$$\therefore d_i = (x_i+1)^2 + y_i^2 - r^2 + (x_i+1)^2 + (y_i-1)^2 - r^2$$

$$d_i = 2(x_i+1)^2 + y_i^2 + (y_i-1)^2 - 2r^2$$

When $d_i < 0$, we have $|D(T)| < |D(S)|$ and pixel T is chosen.

When $d_i \geq 0$, we have $|D(T)| \geq D(S)|$ and pixel S is selected.

Decision variable, d_{i+1} for the next step:

$$d_{i+1} = 2(x_{i+1}+1)^2 + y_{i+1}^2 + (y_{i+1}-1)^2 - 2r^2$$

Hence,

$$d_{i+1} - d_i = 2(x_{i+1}+1)^2 + y_{i+1}^2 + (y_{i+1}-1)^2 - 2r^2 - 2(x_i+1)^2 + y_i^2 + (y_i-1)^2 + 2r^2$$

Since, $x_{i+1} = x_i + 1$

$$d_{i+1} - d_i = 2(x_i+2)^2 + y_{i+1}^2 + (y_{i+1}-1)^2 - 2(x_i+1)^2 - y_i^2 - (y_i-1)^2$$

$$= 2[x_i^2 + 4 + 4x_i - x_i^2 - 1 - 2x_i] + y_{i+1}^2 + y_{i+1}^2 + 1 - 2y_{i+1} - y_i^2 - y_i^2 - 1 + 2y_i$$

$$= 2(2x_i + 3) + 2y_{i+1}^2 - 2y_i^2 - 2y_{i+1} + 2y_i$$

$$= 4x_i + 6 + 2(y_{i+1}^2 - y_i^2) - 2(y_{i+1} - y_i)$$

$$\boxed{d_{i+1} = d_i + 4x_i + 2(y_{i+1}^2 - y_i^2) - 2(y_{i+1} - y_i) + 6}$$

If T is the chosen pixel (that means $d_i < 0$) then $y_{i+1} = y_i$ and so,

$$d_{i+1} = d_i + 4x_i + 6$$

And, if S is the chosen pixel (that means $d_i < 0$) then $y_{i+1} = y_i - 1$ and so,

$$d_{i+1} = d_i + 4(x_i - y_i) + 10$$

$\therefore$ we have,

$$d_{i+1} = \begin{cases} d_i + 4x_i + 6 & \text{if } d_i < 0 \\ d_i + 4(x_i - y_i) + 10 & \text{if } d_i \geq 0 \end{cases}$$

Finally, we set $(0, r)$ to be the starting pixel coordinates and compute the base case value d_1 for this recursive formula from the original definition of d_i:

So,

$$d_1 = 2(0+1)^2 + r^2 + (r-1)^2 - 2r^2$$

$$= 3 - 2r$$

Date: / /

Algorithm for generating all the pixel coordinates in the 90° to 45° octant that are needed when scan-converting a circle of radius r.

```
int x = 0, y = r, d = 3-2r;
while (x ≤ y) {

    setPixel (x, y);
    if (d < 0)
      d = d + 4x + 6;
    else {
      d = d + 4(x-y) + 10;
      y--;
    }
    x++;
}
```

Note:-
During each iteration of the while loop we first set a pixel whose position has already been determined, starting with (0, r). Then, test the current value of decision variable d in order to update d and determine the proper y coordinate of the next pixel. Finally we increment x.

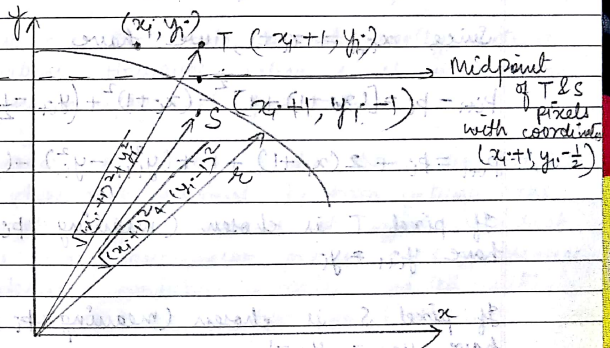
Page no
Date: / /

MIDPOINT CIRCLE ALGORITHM

→ Incremental circle algorithm that is very similar to Bresenham's approach.

→ It is based on the following function for testing the spatial relationship between an arbitrary point (x, y) and a circle of radius r centered at the origin:

$$f(x,y) = x^2 + y^2 - r^2 \begin{cases} <0 & \text{for } (x,y) \text{ inside the circle} \\ =0 & \text{for } (x,y) \text{ on the circle} \\ >0 & \text{for } (x,y) \text{ outside the circle} \end{cases}$$



Choosing Pixel in Midpoint Circle Algorithm

Consider the coordinates of the point halfway between pixel T and pixel S in the above figure $(x_i + 1, y_i - \frac{1}{2})$. This is called midpoint and we use it to define a decision parameter.

$$p_i = f(x_i+1, y_i-\tfrac{1}{2}) = (x_i+1)^2 + (y_i-\tfrac{1}{2})^2 - r^2$$

If p_i is negative, the midpoint is inside the circle, and we choose pixel T.

& If p_i is positive (or equal to zero), the midpoint is outside the circle (or on the circle), and we choose pixel S.

Similarly, the decision parameter for the next step is

$$p_{i+1} = (x_{i+1}+1)^2 + (y_{i+1}-\tfrac{1}{2})^2 - r^2$$

Since, $x_{i+1} = x_i + 1$, we have

$$p_{i+1} - p_i = [(x_i+1)+1]^2 - (x_i+1)^2 + (y_{i+1}-\tfrac{1}{2})^2 - (y_i-\tfrac{1}{2})^2$$

$$\therefore p_{i+1} = p_i + 2(x_i+1) + 1 + (y_{i+1}^2 - y_i^2) - (y_{i+1} - y_i)$$

If pixel T is chosen (meaning $p_i < 0$), we have $y_{i+1} = y_i$

If pixel S is chosen (meaning $p_i \geq 0$), we have $y_{i+1} = y_i - 1$.

$$\therefore p_{i+1} = \begin{cases} p_i + 2(x_i+1) + 1 & \text{if } p_i < 0 \\ p_i + 2(x_i+1) + 1 - 2(y_i - 1) & \text{if } p_i \geq 0 \end{cases}$$

More simplified decision parameter in terms of (x_i, y_i) is :-

$$p_{i+1} = \begin{cases} p_i + 2x_i + 3 & \text{if } p_i < 0 \\ p_i + 2(x_i - y_i) + 5 & \text{if } p_i \geq 0 \end{cases}$$

Or, we can write it in terms of (x_{i+1}, y_{i+1}) :-

$$p_{i+1} = \begin{cases} p_i + 2x_{i+1} + 1 & \text{if } p_i < 0 \\ p_i + 2(x_{i+1} - y_{i+1}) + 1 & \text{if } p_i \geq 0 \end{cases}$$

Finally, we compute the initial value for the decision parameter using the original definition of f_r and $(0, r)$:-

$$p_i = (0+1)^2 + (r - \tfrac{1}{2})^2 - r^2 = \frac{5}{4} - r$$

You can see that this is not really integer computation. However, when r is an integer we can simply set $p_i = 1 - r$.

The error of being $\frac{1}{4}$ less than the precise value does not prevent p_i from getting the appropriate sign. It does not affect the rest of the scan-conversion process either, because the decision variable is only updated with integer increments in subsequent steps.

Page no
Date: / /

Midpoint algorithm that generates the pixel coordinates in the 90° to 45° octant :-

```
int x=0, y=R, p=1-R;
while (x<=y) {
  setPixel(x,y);
  if (p<0)
    p = p+2x+3;
  else {
    p = p + 2(x-y)+5;
    y--;
  }
  x++;
}
```

Note:-

In these discussed circle algorithms, we have assumed that a circle is centered at the origin. To scan-convert a circle centered at (x_c, y_c), we can simply replace the setPixel(x,y) statement in algorithm description with setPixel($x+x_c$, $y+y_c$).

The reason for this to work is that a circle centered at (x_c, y_c), can be viewed as a circle centered at the origin that is moved by x_c and y_c in the x and y direction, respectively.

The effect of scan-converting this arbitrarily centered circle by relocating scan-converted pixels in the same way as moving the circle's center from the origin.

Page no
Date: / /

SCAN CONVERTING AN ELLIPSE

→ Like the circle, ellipse also shows symmetry.

→ In the case of an ellipse, however, symmetry is four-rather than eight way.

→ There are two-ways of mathematically defining an ellipse.
1) Polynomial Method
2) Trigonometric Method

Polynomial Method

The polynomial method of defining an ellipse is given by the expression.

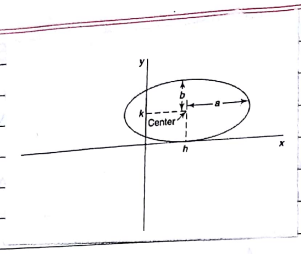
$$\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$$

where, (h,k) = ellipse center
a = length of major axis
b = length of minor axis

When the polynomial method is used to define an ellipse, the value of x is incremented from h to a.
For each step of x, each value of y is found by evaluating the expression.

$$y = b\sqrt{1 - \frac{(x-h)^2}{a^2}} + k$$

Date: / /



Polynomial Description of an Ellipse

This method is very inefficient, however, because the squares of a and $(x-h)$ must be found. Then, floating-point division of $(x-h)^2$ by a^2 and floating-point multiplication of the square root of $[1-(x-h)^2/a^2]$ by b must performed.

Routines have been found that will scan-convert general polynomial equations, including the ellipse. However, these routines are logic intensive and thus are very slow methods for scan-converting ellipses.

Trigonometric Method

The following equations define an ellipse trigonometrically :-

$$x = a\cos\theta + h$$
$$y = b\sin\theta + k$$

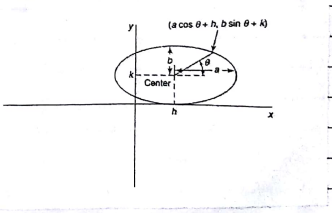
Page no
Date: / /

where (x,y) = the current coordinates
a = length of the major axis
b = length of the minor axis
θ = current angle
(h,k) = center of the ellipse

For generation of an ellipse using the trigonometric method, the value of θ is varied from 0 to $\pi/2$ radians (rad). The remaining points are found by symmetry.

While, this method is also inefficient and thus generally too slow for interactive applications, a lookup table containing the values for $\sin\theta$ and $\cos\theta$ with θ ranging from 0 to $\pi/2$ radians can be used.

This method would have been considered unacceptable at one time because of the relatively high cost of the computer memory used to store the values of θ. However, because of the cost of computer memory has plummeted in recent years, this method is now quite acceptable.



Trigonometric Description of an Ellipse

Page no
Date: / /

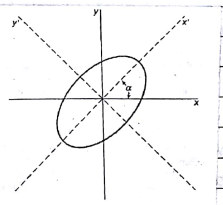
ELLIPSE AXIS ROTATION

Assume that you would like to rotate the ellipse through an angle other than 90°. It can be seen in the following figure that rotation of the ellipse may be accomplished by rotating the x and y axis by α degrees.

When this is done, the equations describing the x, y coordinates of each scan-converted point become

$$x = a\cos\theta - b\sin(\theta+\alpha) + h$$

$$y = b\sin\theta + a\cos(\theta+\alpha) + k$$



Rotation of an ellipse

Page no
Date: / /

Since the ellipse shows four-way symmetry it can easily be rotated by 90°. The new equation is found by trading a and b, the values which describe the major and minor axes.

When the polynomial method is used, the equations used to describe the ellipses become

$$\frac{(x-h)^2}{b^2} + \frac{(y-k)^2}{a^2} = 1$$

where, (h,k) = ellipse center
a = length of major axis
b = length of minor axis

When the trigonometric method is used, the equations used to describe the ellipse become

$$x = b\cos\theta + h$$
$$y = a\sin\theta + k$$

where (x,y) = current coordinates
a = length of major axis
b = length of minor axis
θ = current angle
(h,k) = ellipse center

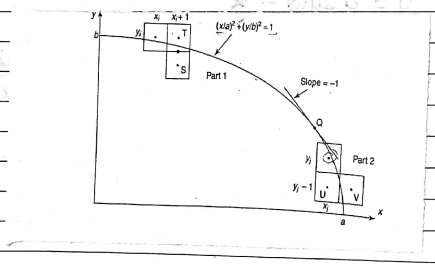
Page no
Date: / /

MIDPOINT ELLIPSE ALGORITHM

→ This is an incremental method for scan-converting an ellipse that is centered at the origin in standard position.

→ Standard position means that the major and minor axes parallel to the coordinate system axes in the ellipse.

→ It works in a way that is very similar to the midpoint circle algorithm. However, because of the four-way symmetry property we need to consider the entire elliptical curve in the first quadrant.



Scan-Converting an Ellipse

The function f that can be used to decide if the midpoint between two candidate pixels is inside or outside the ellipse:-

$$f(x,y) = b^2x^2 + a^2y^2 - a^2b^2 \begin{cases} <0 & (x,y) \text{ inside the ellipse} \\ =0 & (x,y) \text{ on the ellipse} \\ >0 & (x,y) \text{ outside the Ellipse} \end{cases}$$

Divide the elliptical curve from (0,b) to (a,0) into two parts at point Q where the slope of the curve is −1.

Slope of a curve is defined by $f(x,y)=0$ is $\frac{dy}{dx} = -\frac{fx}{fy}$, where fx and fy are partial derivatives of $f(x,y)$ with respect to x and y, respectively.

$$fx = 2b^2x$$
$$fy = 2a^2y$$

$$\frac{dy}{dx} = \frac{-2b^2x}{2a^2y}$$

This shows that the slope of the curve changes monotonically from one side of Q to the other. Hence, we can monitor the slope value during the scan-conversion process to detect Q.

Our starting point is (0,b). Suppose that the coordinates of the last scan-converted pixel upon entering step i are (x_i, y_i). Either $T(x_i+1, y_i)$ or $S(x_i+1, y_i-1)$ to be the next pixel is to be selected.

Date: / /

The midpoint of the vertical line connecting T and S is used to define the following decision parameter :-

$$p_i = f(x_i+1, y_i-\tfrac{1}{2}) = b^2(x_i+1)^2 + a^2(y_i-\tfrac{1}{2})^2 - a^2b^2$$

If $p_i < 0$, the midpoint is inside the curve, and we choose pixel T.

If $p_i \geq 0$, the midpoint is outside (when $p_i > 0$) or on the curve (when equal to zero, $p_i = 0$), and we choose pixel S.

Decision parameter for the next step :-

$$p_{i+1} = f(x_{i+1}, y_{i+1}-\tfrac{1}{2}) = b^2(x_{i+1}+1)^2 + a^2(y_{i+1}-\tfrac{1}{2})^2 - a^2b^2$$

Since, $x_{i+1} = x_i + 1$, we have

$$p_{i+1} - p_i = b^2[(x_{i+1}+1)^2 - x_{(i+1)}^2] + a^2[(y_{i+1}-\tfrac{1}{2})^2 - (y_i-\tfrac{1}{2})^2]$$

$$\therefore p_{i+1} = p_i + 2b^2 + b^2 + a^2[(y_{i+1}-\tfrac{1}{2})^2 - (y_i-\tfrac{1}{2})^2]$$

If T is the chosen pixel (meaning $p_i < 0$), we have $y_{i+1} = y_i$.

If S is the chosen pixel (meaning $p_i \geq 0$), we have $y_{i+1} = y_i - 1$.

Thus we can express p_{i+1} in terms of p_i and (x_{i+1}, y_{i+1})

$$p_{i+1} = \begin{cases} p_i + 2b^2x_{i+1} + b^2 & \text{if } p_i < 0 \\ p_i + 2b^2x_{i+1} + b^2 - 2a^2y_{i+1} & \text{if } p_i \geq 0 \end{cases}$$

Page no
Date: / /

The initial value for this recursive expression can be obtained by evaluating the original definition of p_i with $(0, b)$:-

$$p_1 = b^2 + a^2(b-\tfrac{1}{2})^2 - a^2b^2$$

$$= b^2 - a^2b + \frac{a^2}{4}$$

For Part 2 of curve

Suppose pixel (x_j, y_j) has just been scan-converted upon entering step j. The next pixel is either $U(x_j, y_j-1)$ or $V(x_j+1, y_j-1)$. The midpoint of the horizontal line connecting U and V is used to define the decision parameter

$$q_j = f(x_j+\tfrac{1}{2}, y_j-1) = b^2(x_j+\tfrac{1}{2})^2 + a^2(y_j-1)^2 - a^2b^2$$

If $q_j < 0$, the midpoint is inside the curve and V is chosen.

If $q_j \geq 0$, it is outside or on the curve ($q_j = 0$) and U is chosen.

$$q_{j+1} = f(x_{j+1}+\tfrac{1}{2}, y_{j+1}-1) = b^2(x_{j+1}+\tfrac{1}{2})^2 + a^2(y_{j+1}-1)^2 - a^2b^2$$

Since, $x_{j+1} = x_j + 1$,

$$q_{j+1} - q_j = b^2[(x_{j+1}+\tfrac{1}{2})^2 - (x_j+\tfrac{1}{2})^2] + a^2[(y_{j+1}-1)^2 - y_{j+1}^2]$$

Hence,

$$q_{j+1} = q_j + b^2[(x_{j+1}+\tfrac{1}{2})^2 - (x_j+\tfrac{1}{2})^2] - 2a^2y_{j+1} + a^2$$

If V is chosen pixel (meaning $q_j < 0$, we have $x_{j+1} = x_j + 1$.

If U is chosen pixel (meaning $q_j \geq 0$), we have $x_{j+1} = x_j$.

Thus, we can express q_{j+1} in terms of q_j and (x_{j+1}, y_{j+1}):

$$q_{j+1} = \begin{cases} q_j + 2b^2 x_{j+1} - 2a^2 y_{j+1} + a^2 & \text{if } q_j < 0 \\ q_j - 2a^2 y_{j+1} + a^2 & \text{if } q_j \geq 0 \end{cases}$$

The initial value for this recursive expression is computed using the original definition of q_j and the coordinates (x_k, y_k) of the last pixel chosen for part 1 of the curve:

$$q_1 = f\left(x_k + \frac{1}{2}, y_k - 1\right) = b^2\left(x_k + \frac{1}{2}\right)^2 + a^2(y_k - 1)^2 - a^2b^2$$

Midpoint Algorithm for Scan-Converting the Elliptical Curve in the First Quadrant

⇒ Calculate slope of the curve by evaluating partial derivatives f_x and f_y at each chosen pixel position.

i.e. $f_x = 2b^2 x_i$, $f_y = 2a^2 y_i$ } for position (x_i, y_i)

For Part 1,

$x_{i+1} = x_i + 1$ & $y_{i+1} = y_i$ or $y_i - 1$

For Part 2,

$x_{j+1} = x_j$ or $x_j + 1$ & $y_{j+1} = y_j - 1$

The partial derivatives can be updated incrementally using $2b^2$ and/or $-2a^2$. for example, if $x_{i+1} = x_i + 1$ and $y_{i+1} = y_i - 1$, the partial derivatives for position (x_{i+1}, y_{i+1}) are $2b^2 x_{i+1} = 2b^2 x_i + 2b^2$ and $2a^2 y_{i+1} = 2a^2 y_i - 2a^2$

⇒ Since, $2b^2 x_{i+1}$ and $2a^2 y_{i+1}$ appear in the recursive expression for the decision parameters, and they can be used to efficiently compute p_{i+1} as well as q_{j+1}:-

```
int x = 0, y = b;  // starting point
int aa = a*a, bb = b*b, aa2 = aa*2, bb2 = bb*2;
int fx = 0, fy = aa2*b; // initial partial derivatives
int p = bb - aa*b + 0.25*aa  /*compute & Round off p1 */
```

```
while (fx < fy) { /* |slope| < 1 */
    setPixel(x, y);
    x++;
    fx = fx + bb2;
    if (p<0)
        p = p + fx + bb;
    else {
        y--;
        fy = fy - aa2;
        p = p + fx + bb - fy;
    }
}
setPixel(x, y); /* set pixel at (xk, yk) */
p = bb(x+0.5)(x+0.5) + aa(y-1)(y-1) - aabb
while (y>0) {
    y--;
    fy = fy - aa2;
    if (p>=0)
        p = p - fy + aa;
    else {
        x++;
        fx = fx + bb2;
        p = p + fx - fy + aa;
    }
    setPixel(x, y);
}
```