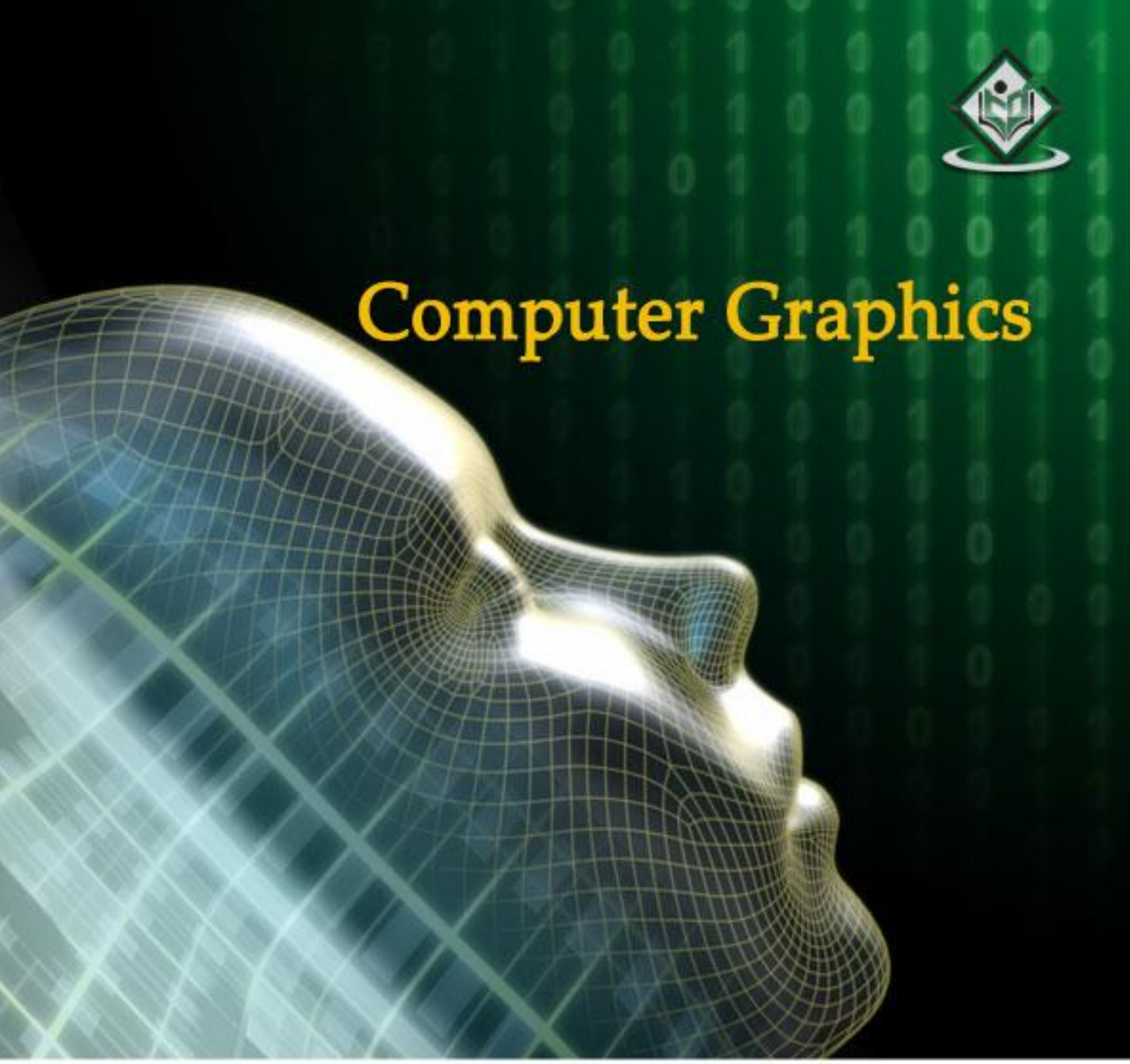


Computer Graphics

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

To display a picture of any size on a computer screen is a difficult process. Computer graphics are used to simplify this process. Various algorithms and techniques are used to generate graphics in computers. This tutorial will help you understand how all these are processed by the computer to give a rich visual experience to the user.

Audience

This tutorial has been prepared for students who don't know how graphics are used in computers. It explains the basics of graphics and how they are implemented in computers to generate various visuals.

Prerequisites

Before you start proceeding with this tutorial, we assume that you are already aware of the basic concepts of C programming language and basic mathematics.

Copyright & Disclaimer

Table of Contents

1. Computer Graphics – Basics

Computer graphics is an art of drawing pictures on computer screens with the help of programming. It involves computations, creation, and manipulation of data. In other words, we can say that computer graphics is a rendering tool for the generation and manipulation of images.

Cathode Ray Tube

The primary output device in a graphical system is the video monitor. The main element of a video monitor is the **Cathode Ray Tube (CRT)**, shown in the following illustration.

The operation of CRT is very simple:

1. The electron gun emits a beam of electrons (cathode rays).
2. The electron beam passes through focusing and deflection systems that direct it towards specified positions on the phosphor-coated screen.
3. When the beam hits the screen, the phosphor emits a small spot of light at each position contacted by the electron beam.
4. It redraws the picture by directing the electron beam back over the same screen points quickly.

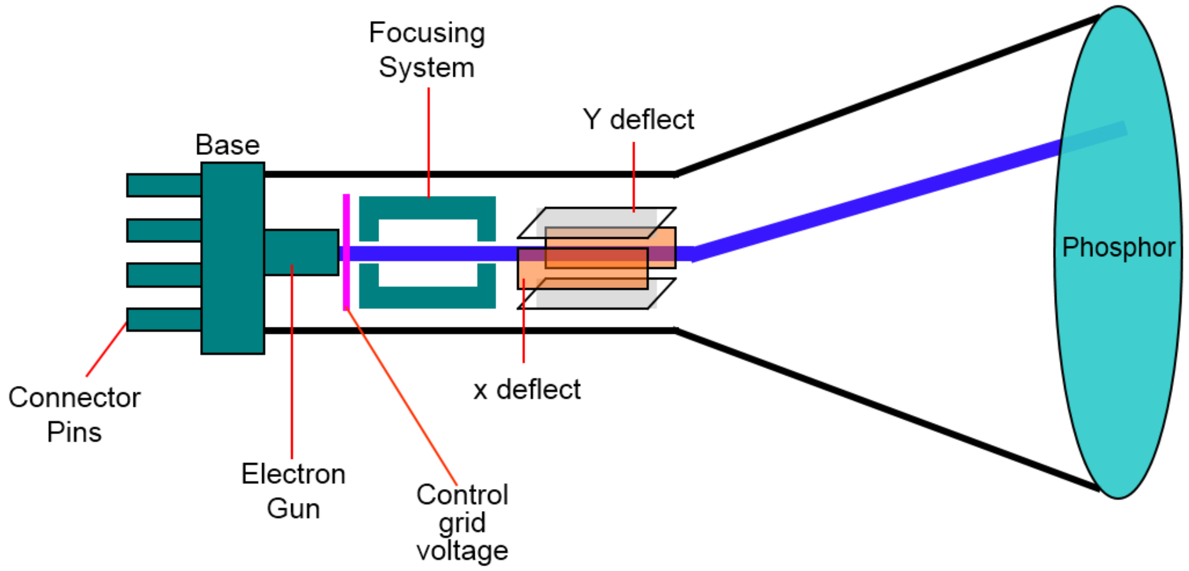


Figure: Cathode Ray Tube

There are two ways (Random scan and Raster scan) by which we can display an object on the screen.

Raster Scan

In a raster scan system, the electron beam is swept across the screen, one row at a time from top to bottom. As the electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots.

Picture definition is stored in memory area called the **Refresh Buffer** or **Frame Buffer**. This memory area holds the set of intensity values for all the screen points. Stored intensity values are then retrieved from the refresh buffer and "painted" on the screen one row (scan line) at a time as shown in the following illustration.

Each screen point is referred to as a **pixel (picture element)** or **pel**. At the end of each scan line, the electron beam returns to the left side of the screen to begin displaying the next scan line.

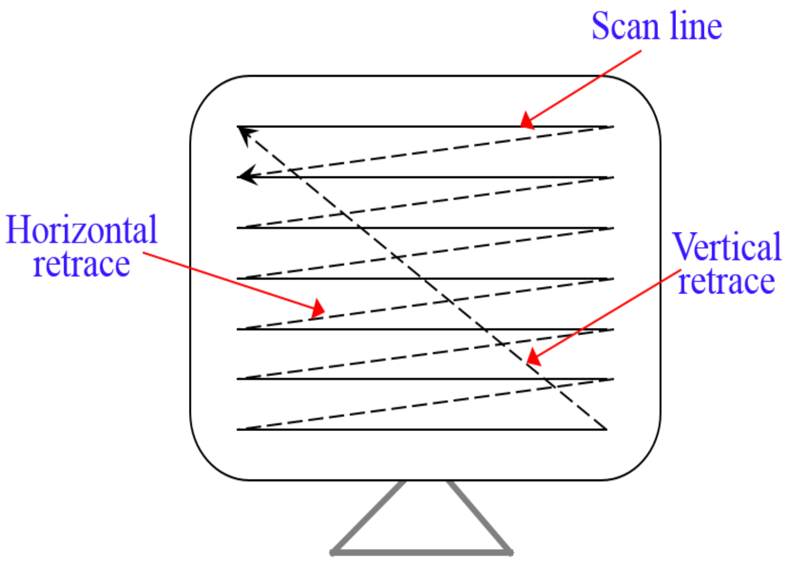


Figure: Raster Scan

Random Scan (Vector Scan)

In this technique, the electron beam is directed only to the part of the screen where the picture is to be drawn rather than scanning from left to right and top to bottom as in raster scan. It is also called **vector display**, **stroke-writing display**, or **calligraphic display**.

Picture definition is stored as a set of line-drawing commands in an area of memory referred to as the **refresh display file**. To display a specified picture, the system cycles through the set of commands in the display file, drawing each component line in turn. After all the line-drawing commands are processed, the system cycles back to the first line command in the list.

Random-scan displays are designed to draw all the component lines of a picture 30 to 60 times each second.

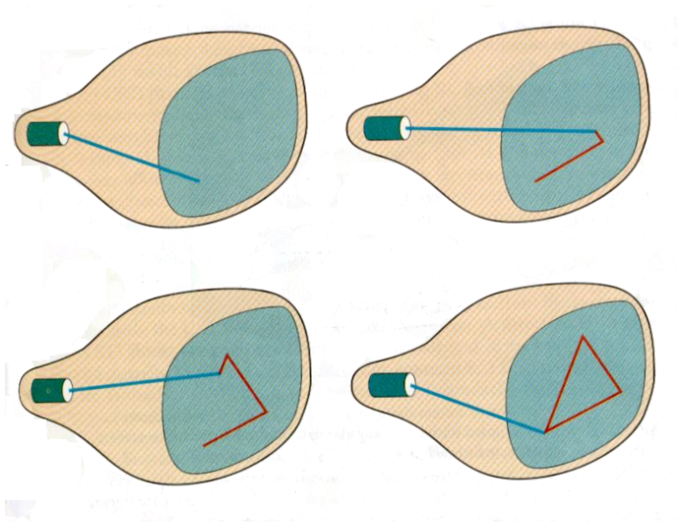


Figure: Random Scan

Application of Computer Graphics

Computer Graphics has numerous applications, some of which are listed below:

- **Computer graphics user interfaces** (GUIs) – A graphic, mouse-oriented paradigm which allows the user to interact with a computer.
- **Business presentation graphics** - "A picture is worth a thousand words".
- **Cartography** - Drawing maps.
- **Weather Maps** – Real-time mapping, symbolic representations.
- **Satellite Imaging** - Geodesic images.
- **Photo Enhancement** - Sharpening blurred photos.
- **Medical imaging** - MRIs, CAT scans, etc. - Non-invasive internal examination.

- **Engineering drawings** - mechanical, electrical, civil, etc. - Replacing the blueprints of the past.

- **Typography** - The use of character images in publishing - replacing the hard type of the past.

- **Architecture** - Construction plans, exterior sketches - replacing the blueprints and hand drawings of the past.

- **Art** - Computers provide a new medium for artists.

- **Training** - Flight simulators, computer aided instruction, etc.

- **Entertainment** - Movies and games.

- **Simulation and modeling** - Replacing physical modeling and enactments

2. Computer Graphics – Line Generation Algorithm

A line connects two points. It is a basic element in graphics. To draw a line, you need two points between which you can draw a line. In the following three algorithms, we refer the one point of line as X_0, Y_0 and the second point of line as X_1, Y_1.

DDA Algorithm

Digital Differential Analyzer (DDA) algorithm is the simple line generation algorithm which is explained step by step here.

Step 1: Get the input of two end points (X_0, Y_0) and (X_1, Y_1).

Step 2: Calculate the difference between two end points.

```
             dx = X₁ - X₀
            dy = Y₁ - Y₀
```

Step 3: Based on the calculated difference in step-2, you need to identify the number of steps to put pixel. If dx > dy, then you need more steps in x coordinate; otherwise in y coordinate.

```
     if(absolute(dx) > absolute(dy))
            Steps = absolute(dx);
     else
            Steps = absolute(dy);
```

Step 4: Calculate the increment in x coordinate and y coordinate.

```
     Xincrement = dx / (float) steps;
     Yincrement = dy / (float) steps;
```

Step 5: Put the pixel by successfully incrementing x and y coordinates accordingly and complete the drawing of the line.

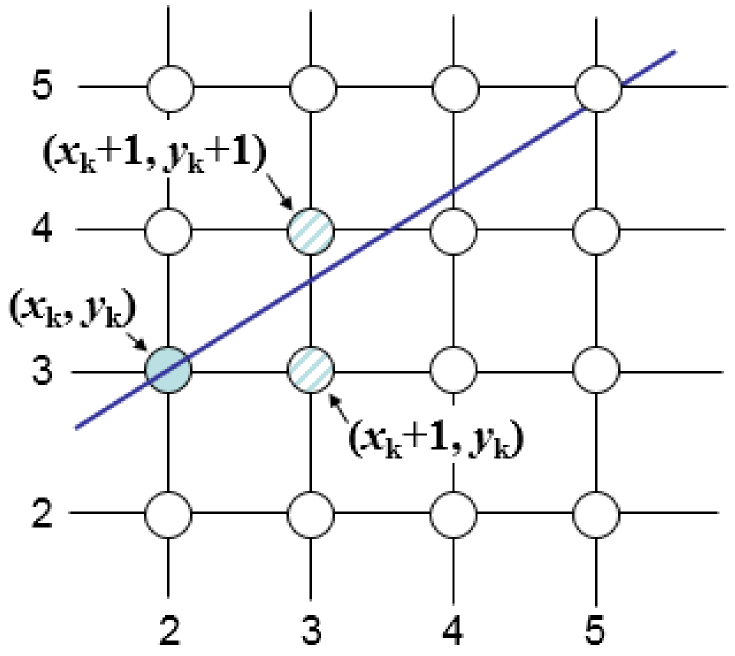
```
for(int v=0; v < Steps; v++)
{
     x = x + Xincrement;
```

```
    y = y + Yincrement;
    putpixel(Round(x), Round(y));
}
```

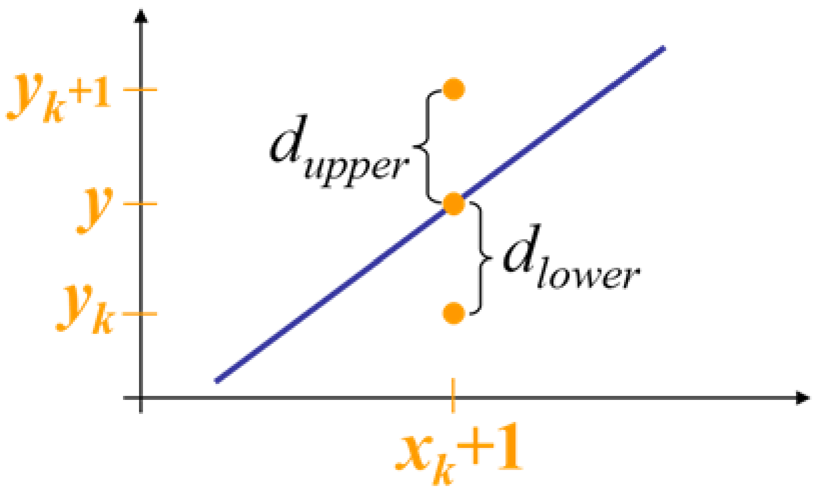
Bresenham's Line Generation

The Bresenham algorithm is another incremental scan conversion algorithm. The big advantage of this algorithm is that, it uses only integer calculations. Moving across the *x* axis in unit intervals and at each step choose between two different *y* coordinates.

For example, as shown in the following illustration, from position (2, 3) you need to choose between (3, 3) and (3, 4). You would like the point that is closer to the original line.



At sample position x_k+1, the vertical separations from the mathematical line are labelled as d_{upper} and d_{lower}.



From the above illustration, the *y* coordinate on the mathematical line at x_k+1 is:

$$Y = m(X_k + 1) + b$$

So, d_{upper} and d_{lower} are given as follows:

$$\begin{aligned} d_{lower} &= y - y_k \\ &= m(X_k + 1) + b - Y_k \end{aligned}$$

and

$$\begin{aligned} d_{upper} &= (y_k + 1) - y \\ &= Y_k + 1 - m(X_k + 1) - b \end{aligned}$$

You can use these to make a simple decision about which pixel is closer to the mathematical line. This simple decision is based on the difference between the two pixel positions.

$$d_{lower} - d_{upper} = 2m(x_k + 1) - 2y_k + 2b - 1$$

Let us substitute *m* with dy/dx where dx and *dy* are the differences between the end-points.

$$\begin{aligned} dx(d_{lower} - d_{upper}) &= dx(2\tfrac{dy}{dx}(x_k + 1) - 2y_k + 2b - 1) \\ &= 2dy \cdot x_k - 2dx \cdot y_k + 2dy + dx(2b - 1) \\ &= 2dy \cdot x_k - 2dx \cdot y_k + C \end{aligned}$$

So, a decision parameter p_k for the *k*th step along a line is given by:

$$p_k = dx(d_{lower} - d_{upper})$$

$$= 2dy \cdot x_k - 2dx \cdot y_k + C$$

The sign of the decision parameter p_k is the same as that of $d_{lower} - d_{upper}$.

If p_k is negative, then choose the lower pixel, otherwise choose the upper pixel.

Remember, the coordinate changes occur along the *x* axis in unit steps, so you can do everything with integer calculations. At step *k*+1, the decision parameter is given as:

$$p_{k+1} = 2dy \cdot x_{k+1} - 2dx \cdot y_{k+1} + C$$

Subtracting p_k from this we get:

$$p_{k+1} - p_k = 2dy(x_{k+1} - x_k) - 2dx(y_{k+1} - y_k)$$

But, x_{k+1} is the same as x_k+1. So:

$$p_{k+1} = p_k + 2dy - 2dx(y_{k+1} - y_k)$$

Where, $y_{k+1} - y_k$ is either 0 or 1 depending on the sign of p_k.

The first decision parameter p0 is evaluated at (x0, y0) is given as:

$$p_0 = 2dy - dx$$

Now, keeping in mind all the above points and calculations, here is the Bresenham algorithm for slope m < 1:

Step 1: Input the two end-points of line, storing the left end-point in (x_0, y_0).

Step 2: Plot the point (x_0, y_0).

Step 3: Calculate the constants d*x*, d*y*, 2d*y*, and (2d*y* – 2d*x*) and get the first value for the decision parameter as:

$$p_0 = 2dy - dx$$

Step 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

$p_{k+1} = p_k + 2dy$Otherwise, the next point to plot is (x_k+1, y_k+1) and

$$p_{k+1} = p_k + 2dy - 2dx$$

Step 5: Repeat step 4 (d*x* – 1) times.

For m > 1, find out whether you need to increment x while incrementing y each time. After solving, the equation for decision parameter p_k will be very similar, just the x and y in the equation gets interchanged.

Mid-Point Algorithm

Mid-point algorithm is due to Bresenham which was modified by Pitteway and Van Aken. Assume that you have already put the point P at (x, y) coordinate and the slope of the line is $0 \leq k \leq 1$ as shown in the following illustration.

Now you need to decide whether to put the next point at E or N. This can be chosen by identifying the intersection point Q closest to the point N or E. If the intersection point Q is closest to the point N then N is considered as the next point; otherwise E.

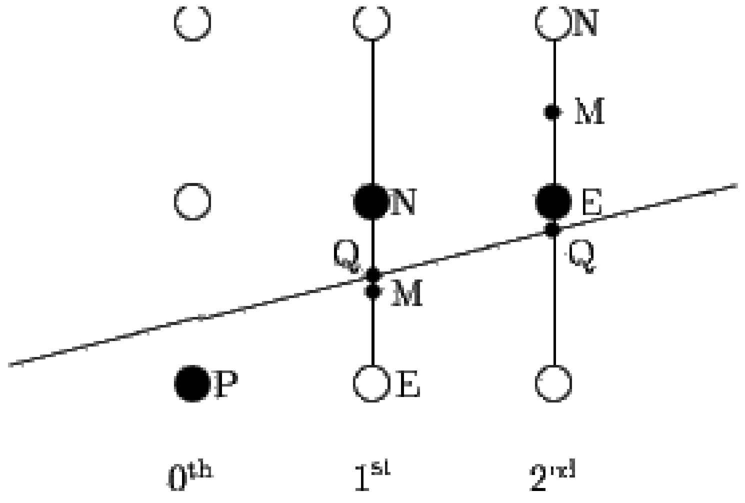


Figure: Mid-point Algorithm

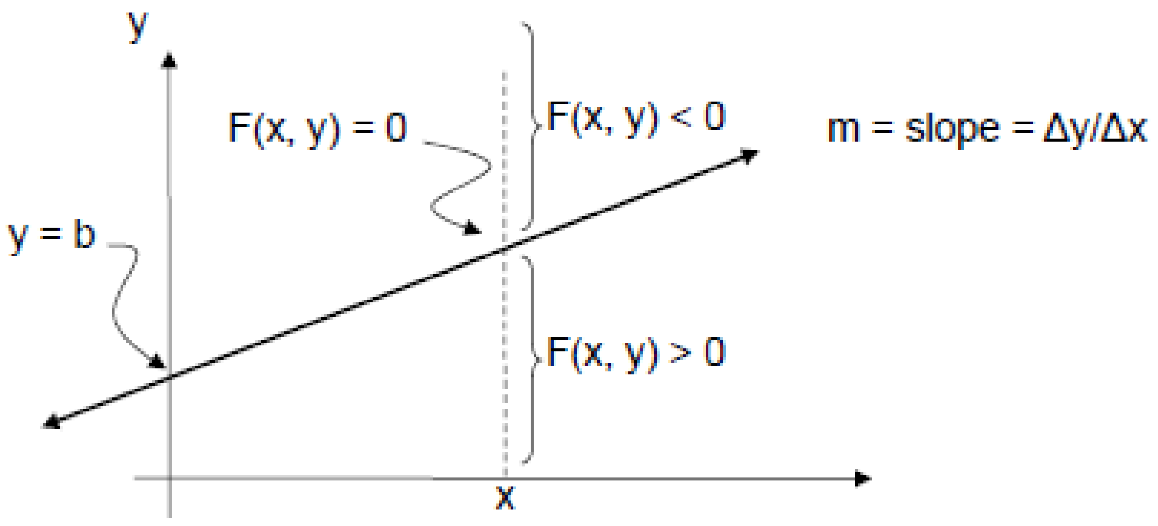
To determine that, first calculate the mid-point M(x+1, y + ½). If the intersection point Q of the line with the vertical line connecting E and N is below M, then take E as the next point; otherwise take N as the next point.

In order to check this, we need to consider the implicit equation:

$$F(x,y) = mx + b - y$$

For positive m at any given X,

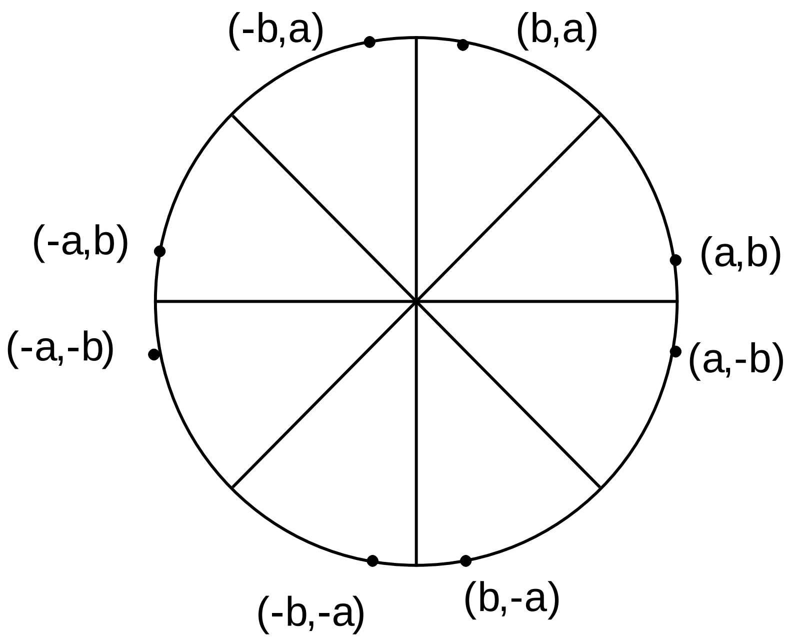
- If y is on the line, then $F(x, y) = 0$
- If y is above the line, then $F(x, y) < 0$
- If y is below the line, then $F(x, y) > 0$


y
F(x, y) = 0
F(x, y) < 0
m = slope = Δy/Δx
y = b
F(x, y) > 0
x

3. Computer Graphics – Circle Generation Algorithm

Drawing a circle on the screen is a little complex than drawing a line. There are two popular algorithms for generating a circle: **Bresenham's Algorithm** and **Midpoint Circle Algorithm.** These algorithms are based on the idea of determining the subsequent points required to draw the circle. Let us discuss the algorithms in detail:

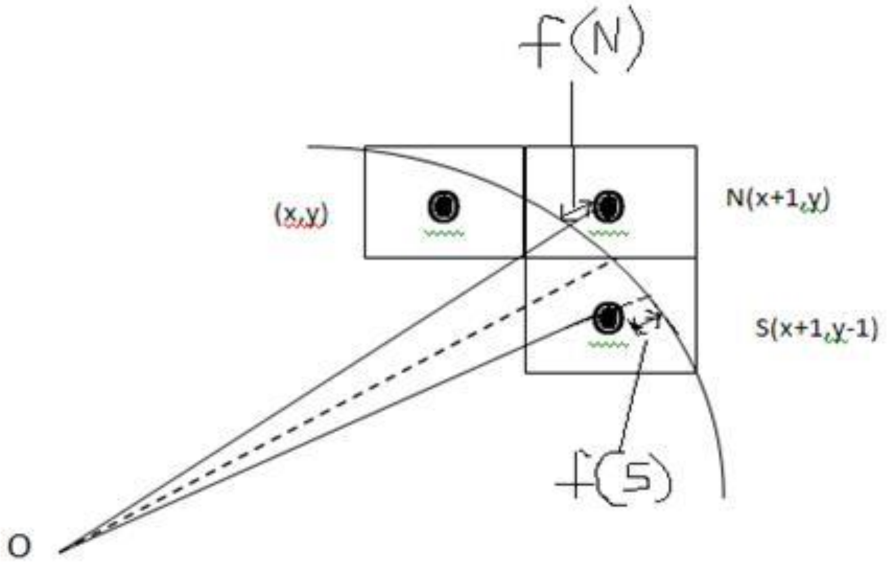
The equation of circle is $X^2 + Y^2 = r^2$, where r is radius.



Bresenham's Algorithm

We cannot display a continuous arc on the raster display. Instead, we have to choose the nearest pixel position to complete the arc.

From the following illustration, you can see that we have put the pixel at (X, Y) location and now need to decide where to put the next pixel: at N (X+1, Y) or at S (X+1, Y-1).



This can be decided by the decision parameter **d**.

- If d <= 0, then N(X+1, Y) is to be chosen as next pixel.
- If d > 0, then S(X+1, Y-1) is to be chosen as the next pixel.

Algorithm

Step 1: Get the coordinates of the center of the circle and radius, and store them in x, y, and R respectively. Set P=0 and Q=R.

Step 2: Set decision parameter D = 3 – 2R.

Step 3: Repeat through step-8 while X < Y.

Step 4: Call Draw Circle (X, Y, P, Q).

Step 5: Increment the value of P.

Step 6: If D < 0 then D = D + 4x + 6.

Step 7: Else Set Y = Y + 1, D = D + 4(X-Y) + 10.

Step 8: Call Draw Circle (X, Y, P, Q).

```
Draw Circle Method(X, Y, P, Q).
Call Putpixel (X + P, Y + Q).
Call Putpixel (X - P, Y + Q).
Call Putpixel (X + P, Y - Q).
Call Putpixel (X - P, Y - Q).
Call Putpixel (X + Q, Y + X).
Call Putpixel (X - Q, Y + X).
Call Putpixel (X + Q, Y - X).
Call Putpixel (X - Q, Y - X).
```

Mid Point Algorithm

Step 1: Input radius **r** and circle center (x_c, y_c) and obtain the first point on the circumference of the circle centered on the origin as

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

Step 2: Calculate the initial value of decision parameter as

$P_0 = 5/4 – r$ (See the following description for simplification of this equation.)

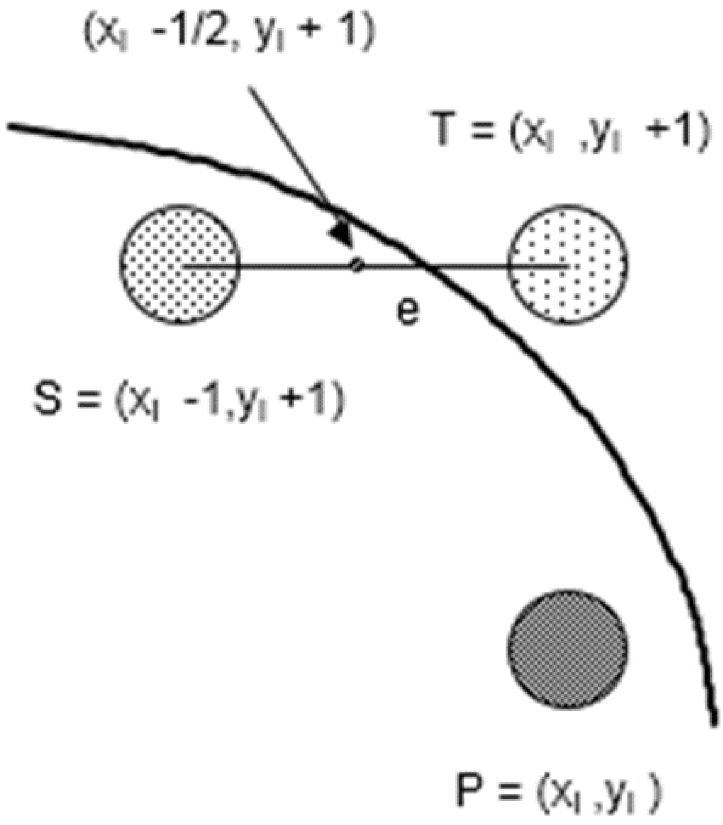
$$f(x, y) = x^2 + y^2 - r^2 = 0$$

$$f(x_i - 1/2 + e, y_i + 1)$$

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

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

$$= f(x_i - 1/2, y_i + 1) + 2(x_i - 1/2)e + e^2 = 0$$



```
Let di = f(xᵢ - 1/2, yᵢ + 1) =  -2(xᵢ - 1/2)e - e²
Thus,
If e < 0 then di > 0 so choose point S = (xᵢ - 1, yᵢ + 1).
dᵢ₊₁  = f(xᵢ – 1 - 1/2, yᵢ + 1 + 1) = ((xᵢ - 1/2) - 1)² + ((yᵢ + 1) + 1)² - r²
            = dᵢ - 2(xᵢ - 1) + 2(yᵢ + 1) + 1
            = dᵢ + 2(yᵢ₊₁ - xᵢ₊₁) + 1

If e >= 0 then dᵢ <= 0 so choose point T = (xᵢ, yᵢ + 1)
    dᵢ₊₁      = f(xᵢ - 1/2, yᵢ + 1 + 1)
            = dᵢ + 2yᵢ₊₁ + 1

The initial value of dᵢ is
      d₀    =  f(r - 1/2,  0 + 1) = (r - 1/2)₂ + 1² - r²
            =  5/4 - r   {1 - r can be used if r is an integer}
```

```
When point S = ($x_i$ - 1, $y_i$ + 1) is chosen, then
     $d_{i+1}$   = $d_i$ + $2x_{i+1}$ + $2y_{i+1}$ + 1

When point T = ($x_i$, $y_i$ + 1) is chosen then
     $d_{i+1}$ = $d_i$ + $2y_{i+1}$ + 1
```

Step 3: At each X_K position starting at K=0, perform the following test:

```
     If $P_K$ < 0 then next point on circle (0,0) is ($X_{K+1}$,$Y_K$) and
            $P_{K+1}$ = $P_K$ + $2X_{K+1}$ + 1
     Else
            $P_{K+1}$ = $P_K$ + $2X_{K+1}$ + 1 – $2Y_{K+1}$
Where, $2X_{K+1}$ = $2X_{K+2}$ and $2Y_{K+1}$ = $2Y_{K-2}$.
```

Step 4: Determine the symmetry points in other seven octants.

Step 5: Move each calculate pixel position (X, Y) onto the circular path centered on (X_C, Y_C) and plot the coordinate values.

$$X = X + X_C, \qquad Y = Y + Y_C$$

Step 6: Repeat step-3 through 5 until X >= Y.