UNIT 2:

Raster Graphics Algorithms

Simple Raster Graphics Package(SRGP)

A Simple Raster Graphics Package (SRGP) is a basic and straightforward software library or package used in computer graphics to create, manipulate, and display raster graphics. Raster graphics, also known as bitmap graphics, consist of a grid of pixels, where each pixel represents a single point of color in an image. SRGP provides a minimal set of functions and data structures to work with pixel-based images, making it a suitable choice for learning computer graphics fundamentals or implementing simple graphical applications. Here's an overview of the components typically found in a Simple Raster Graphics Package:

1. **Pixel Data Representation:** At the core of an SRGP, there is a data structure to represent the pixel data. This data structure usually consists of a 2D array or a 1D array representing the image's pixels. Each pixel in the array holds information about its color (RGB values) and other attributes.
2. **Canvas Functions:** SRGP provides basic functions to manipulate the canvas (pixel array) directly. These functions allow setting pixel colors, reading pixel colors, and clearing the canvas to a particular color.
3. **Drawing Primitives:** SRGP typically supports basic drawing primitives, such as lines, circles, rectangles, and points. These primitives allow users to draw simple shapes on the canvas.
4. **Color Management:** SRGP provides functions for defining and using colors in the image. It may include functions to set a pixel's color using RGB values, or predefined color names.
5. **Coordinate System:** An SRGP usually defines a coordinate system for the canvas. Users can specify pixel positions using coordinates (x, y), with (0,0) usually located at the top-left corner of the canvas.

6. **Screen Refreshing:** To display the image on the screen, an SRGP needs to refresh the screen periodically. It typically includes a function to update the screen and show the current canvas content.
7. **Keyboard and Mouse Interaction:** In some SRGP implementations, basic keyboard and mouse interaction functions might be provided. These functions allow users to interact with the canvas by reading input from the keyboard or mouse.
8. **Error Handling:** An SRGP might include error handling mechanisms to report and handle errors that occur during graphics operations.

While an SRGP is useful for beginners or for quick prototyping of simple graphical applications, it lacks many advanced features found in full-fledged graphics libraries. More comprehensive graphics libraries like OpenGL, DirectX, or HTML5 Canvas offer a broader range of functionalities, including 3D graphics support, hardware acceleration, and more advanced rendering techniques. However, SRGP can serve as a stepping stone for those learning computer graphics concepts before moving on to more complex graphics frameworks.

Line Drawing

Line Drawing Algorithm

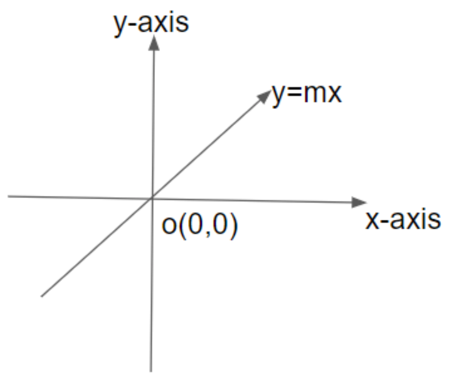
The Line Drawing Algorithm is a graphical algorithm for representing line segments on discrete graphical media, such as printers and pixel-based media."

A line drawing algorithm is a method for estimating a line segment on discrete graphical media such as pixel-based screens and printers in computer graphics. Line sketching on such media necessitates an approximation (in nontrivial cases). Lines are rasterise in one colour using basic methods. Spatial anti-aliasing is a sophisticated approach that allows for a better representation of numerous colour gradations.
To draw a line on a continuous medium, however, no algorithm is require. Cathode-ray oscilloscopes, for example, use analogue phenomena to create lines and curves.

The formula for a slope line interception is:

Y = mx + b

In this formula, m is the slope line and b is the line's intercept of y. Two endpoints for the line segment are supplied in coordinates (x1, y1) and (x2, y2).



Properties of a Line Drawing Algorithm

These Algorithm has the following characteristics.

- **Input:** At least one or more inputs must be accept a good algorithm.
- **Output:** At least one output must produced an algorithm.
- **An algorithm should be precise:** The algorithm's each step must well-define.
- **Finiteness:** Finiteness is require in an algorithm. It signifies that the algorithm will come to a halt once all of the steps have been complete.
- **Correctness:** An algorithm must implemented correctly.
- **Uniqueness:** The result of an algorithm should be based on the given input, and all steps of the algorithm should be clearly and uniquely defined.
- **Effectiveness:** An algorithm's steps must be correct and efficient.
- **Easy to understand:** Learners must be able to understand the solution in a more natural way thanks to an algorithm.

Types of Line Drawing Algorithm

For drawing a line, the following algorithms are use:

- DDA (Digital Differential Analyzer) Line Drawing Algorithm
- Bresenham's Line Drawing Algorithm

DDA Algorithm

DDA stands for Digital Differential Analyzer. It is an incremental method of scan conversion of line. In this method calculation is performed at each step but by using results of previous steps.

Suppose at step i, the pixels is (x_i,y_i)

The line of equation for step i

$y_i=mx_{i+b}$......................equation 1

Next value will be

$y_{i+1}=m_{xi+1}+b$……………..equation 2

$m = \frac{\Delta y}{\Delta x}$

$y_{i+1}-y_i=\Delta y$…………………..equation 3

$y_{i+1}-x_i=\Delta x$………………….equation 4

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

$\Delta y=m\Delta x$

$y_{i+1}=y_i+m\Delta x$

$\Delta x=\Delta y/m$

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

$x_{i+1}=x_i+\Delta y/m$

Case1: When |M|<1 then (assume that $x_1<x_2$)

x= x1,y=y1 set Δx=1

yi+1=y1+m,   x=x+1

Until x = x2</x

Case2: When |M|<1 then (assume that y1<y2)

x= x1,y=y1 set Δy=1

xi+1=$\frac{1}{m}$,   y=y+1

Until y → y2</y

Advantage:

1. It is a faster method than method of using direct use of line equation.
2. This method does not use multiplication theorem.
3. It allows us to detect the change in the value of x and y ,so plotting of same point twice is not possible.
4. This method gives overflow indication when a point is repositioned.
5. It is an easy method because each step involves just two additions.

Disadvantage:

1. It involves floating point additions rounding off is done. Accumulations of round off error cause accumulation of error.
2. Rounding off operations and floating point operations consumes a lot of time.
3. It is more suitable for generating line using the software. But it is less suited for hardware implementation.

DDA Algorithm:

```
Step1: Start Algorithm

Step2: Declare x1,y1,x2,y2,dx,dy,x,y as integer variables.

Step3: Enter value of x1,y1,x2,y2.

Step4: Calculate dx = x2-x1

Step5: Calculate dy = y2-y1

Step6: If ABS (dx) > ABS (dy)
           Then step = abs (dx)
           Else

Step7: xinc=dx/step
           yinc=dy/step
           assign x = x1
           assign y = y1

Step8: Set pixel (x, y)

Step9: x = x + xinc
           y = y + yinc
           Set pixels (Round (x), Round (y))

Step10: Repeat step 9 until x = x2
```

Step11: End Algorithm

Example: If a line is drawn from (2, 3) to (6, 15) with use of DDA. How many points will needed to generate such line?

Solution: P1 (2,3)   P11 (6,15)

x1=2

y1=3

x2= 6

y2=15

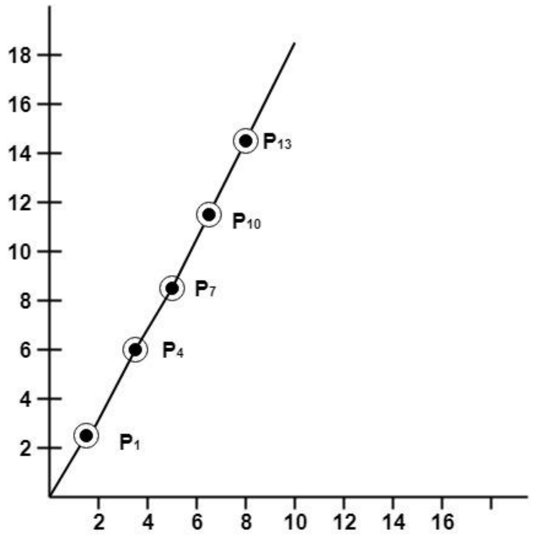
dx = 6 - 2 = 4

dy = 15 - 3 = 12

m = $\frac{dy}{dx} = \frac{12}{4}$

For calculating next value of x takes x = x + $\frac{1}{m}$

$P_1(2, 3)$	point plotted
$P_2(2\frac{1}{3}, 4)$	point plotted
$P_3(2\frac{2}{3}, 5)$	point not plotted
$P_4(3, 6)$	point plotted
$P_5(3\frac{1}{3}, 7)$	point not plotted
$P_6(3\frac{2}{3}, 8)$	point not plotted
$P_7(4, 9)$	point plotted
$P_8(4\frac{1}{3}, 10)$	point not plotted
$P_9(4\frac{2}{3}, 11)$	point not plotted
$P_{10}(5, 12)$	point plotted
$P_{11}(5\frac{1}{3}, 13)$	point not plotted
$P_{12}(5\frac{2}{3}, 14)$	point not plotted
$P_{13}(6, 15)$	point plotted



Bresenham's Line Algorithm

This algorithm is used for scan converting a line. It was developed by Bresenham. It is an efficient method because it involves only integer addition, subtractions, and multiplication operations. These operations can be performed very rapidly so lines can be generated quickly.

In this method, next pixel selected is that one who has the least distance from true line.

The method works as follows:

Assume a pixel $P_1'(x_1',y_1')$,then select subsequent pixels as we work our may to the night, one pixel position at a time in the horizontal direction toward $P_2'(x_2',y_2')$.

Once a pixel in choose at any step

The next pixel is

1. Either the one to its right (lower-bound for the line)
2. One top its right and up (upper-bound for the line)

The line is best approximated by those pixels that fall the least distance from the path between P_1',P_2'.

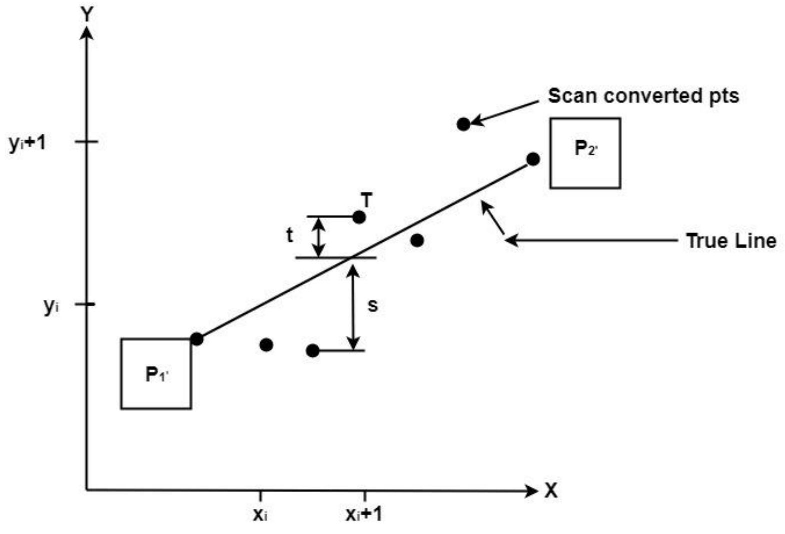


Fig: Scan Converting a line.

To chooses the next one between the bottom pixel S and 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 $x_{i+1}=x_i+1$ and $y_{i+1}=y_i+1$

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

$y=mx_{i+1}+b$

$$\boxed{y = m(x_i + 1) + b}$$

The distance from S to the actual line in y direction

$s = y-y_i$

The distance from T to the actual line in y direction

$t = (y_i+1)-y$

Now consider the difference between these 2 distance values

$s - t$

When $(s-t) <0 \Longrightarrow s < t$

The closest pixel is S

When $(s-t) \geq 0 \Longrightarrow s < t$

The closest pixel is T

This difference is

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

$= 2y - 2yi -1$

$$\boxed{s - t = 2m(x_i + 1) + 2b - 2y_i - 1}$$ [Putting the value of (1)]

Substituting m by $\frac{\triangle y}{\triangle x}$ and introducing decision variable

$d_i=\triangle x\ (s-t)$

$d_i=\triangle x\ (2 \frac{\triangle y}{\triangle x} (x_i+1)+2b-2y_i-1)$

$=2\triangle xy_i-2\triangle y-1\triangle x.2b-2y_i\triangle x-\triangle x$

$d_i=2\triangle y.x_i-2\triangle x.y_i+c$

Where c= $2\triangle y+\triangle x\,(2b-1)$

We can write the decision variable d_{i+1} for the next slip on

$d_{i+1}=2\triangle y.x_{i+1}-2\triangle x.y_{i+1}+c$

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

Since x_(i+1)=x_i+1,we have

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

Special Cases

If chosen pixel is at the top pixel T (i.e., $d_i \geq 0$)$\Longrightarrow y_{i+1}=y_i+1$

$d_{i+1}=d_i+2\triangle y-2\triangle x$

If chosen pixel is at the bottom pixel T (i.e., $d_i<0$)$\Longrightarrow y_{i+1}=y_i$

$d_{i+1}=d_i+2\triangle y$

Finally, we calculate d_1

$d_1=\triangle x[2m(x_1+1)+2b-2y_1-1]$

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

Since $mx_1+b-y_i=0$ and $m = \frac{\triangle y}{\triangle x}$, we have

$d_1=2\triangle y-\triangle x$

Advantage:

1. It involves only integer arithmetic, so it is simple.

2. It avoids the generation of duplicate points.

3. It can be implemented using hardware because it does not use multiplication and division.

4. It is faster as compared to DDA (Digital Differential Analyzer) because it does not involve floating point calculations like DDA Algorithm.

Disadvantage:

1. This algorithm is meant for basic line drawing only Initializing is not a part of Bresenham's line algorithm. So to draw smooth lines, you should want to look into a different algorithm.

Bresenham's Line Algorithm:

```
Step1: Start Algorithm

Step2: Declare variable $x_1,x_2,y_1,y_2,d,i_1,i_2,dx,dy$

Step3: Enter value of $x_1,y_1,x_2,y_2$
        Where $x_1,y_1$ are coordinates of starting point
        And $x_2,y_2$ are coordinates of Ending point

Step4: Calculate $dx = x_2-x_1$
        Calculate $dy = y_2-y_1$
        Calculate $i_1=2*dy$
        Calculate $i_2=2*(dy-dx)$
        Calculate $d=i_1-dx$

Step5: Consider (x, y) as starting point and $x_{end}$ as maximum possible value of x.
        If $dx < 0$
            Then $x = x_2$
            $y = y_2$
             $x_{end}=x_1$
        If $dx > 0$
          Then $x = x_1$
        $y = y_1$
            $x_{end}=x_2$

Step6: Generate point at (x,y)coordinates.

Step7: Check if whole line is generated.
        If $x > = x_{end}$
        Stop.

Step8: Calculate co-ordinates of the next pixel
        If $d < 0$
          Then $d = d + i_1$
        If $d \geq 0$
     Then $d = d + i_2$
        Increment $y = y + 1$

Step9: Increment $x = x + 1$

Step10: Draw a point of latest (x, y) coordinates

Step11: Go to step 7
```

Step12: End of Algorithm

Example: Starting and Ending position of the line are (1, 1) and (8, 5). Find intermediate points.

Solution: $x_1=1$

$y_1=1$

$x_2=8$

$y_2=5$

$dx= x_2-x_1=8-1=7$

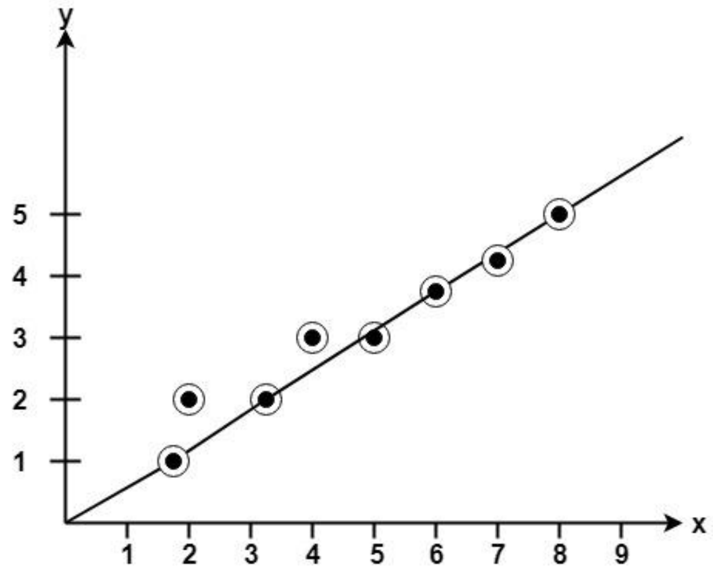
$dy=y_2-y_1=5-1=4$

$I_1=2^* \Delta y=2^*4=8$

$I_2=2^*(\Delta y-\Delta x)=2^*(4-7)=-6$

$d = I_1-\Delta x=8-7=1$

x	**y**	**d=d+I_1 or I_2**
1	1	$d+I_2=1+(-6)=-5$
2	2	$d+I_1=-5+8=3$
3	2	$d+I_2=3+(-6)=-3$
4	3	$d+I_1=-3+8=5$
5	3	$d+I_2=5+(-6)=-1$
6	4	$d+I_1=-1+8=7$
7	4	$d+I_2=7+(-6)=1$
8	5	



Midpoint Line Algorithm

The Midpoint Line Algorithm, also known as Bresenham's Line Algorithm, is a widely used method for rasterizing and drawing lines on a pixel grid in computer graphics. The algorithm is efficient and straightforward, utilizing integer arithmetic to determine which pixels to color in order to approximate the line. It is especially useful for drawing lines on devices with limited resources or without hardware support for line drawing.

Here are the basic concepts of the Midpoint Line Algorithm:

1. **Initial Setup:** To draw a line between two points (x0, y0) and (x1, y1), the algorithm determines the initial point (x, y) to start the line. The algorithm chooses either (x0, y0) or (x1, y1) as the starting point, depending on the slope of the line. The slope is defined as the change in y divided by the change in x.
2. **Decision Parameter:** The Midpoint Line Algorithm calculates an initial decision parameter, often denoted as P0, based on the chosen starting point (x, y) and the endpoints (x0, y0) and (x1, y1). The decision parameter is used to decide which pixel to color at each step along the line.
3. **Incremental Approach:** The algorithm then uses an incremental approach to determine the next pixel to color as it progresses from the starting point towards the endpoint. At each step, it evaluates the decision parameter and updates the coordinates accordingly.
4. **Bresenham's Incremental Method:** Bresenham's incremental method is used to increment the decision parameter and update the coordinates for the next pixel. This

method utilizes integer arithmetic, avoiding the need for floating-point calculations, making it efficient for devices that lack floating-point hardware support.

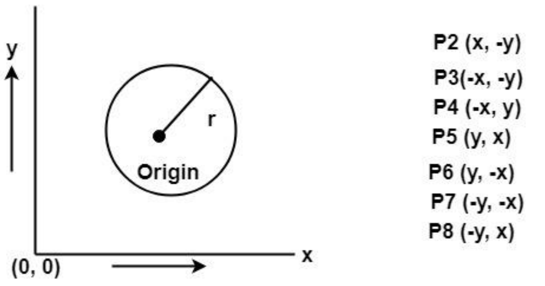
5. **Pixel Plotting:** At each step, the algorithm plots the current pixel (x, y) on the pixel grid, coloring it to approximate the line. Since the algorithm uses integer coordinates, it may introduce slight rounding errors, but the result is a visually close approximation of the desired line.
6. **Steep and Shallow Lines:** The algorithm can handle both steep (|slope| > 1) and shallow (|slope| < 1) lines by selecting the appropriate initial point and adjusting the incremental steps accordingly.
7. **Rasterization and Clipping:** Once the line points are determined by the algorithm, they need to be rasterized onto the display screen, taking into account clipping to ensure that only visible parts of the line are drawn.

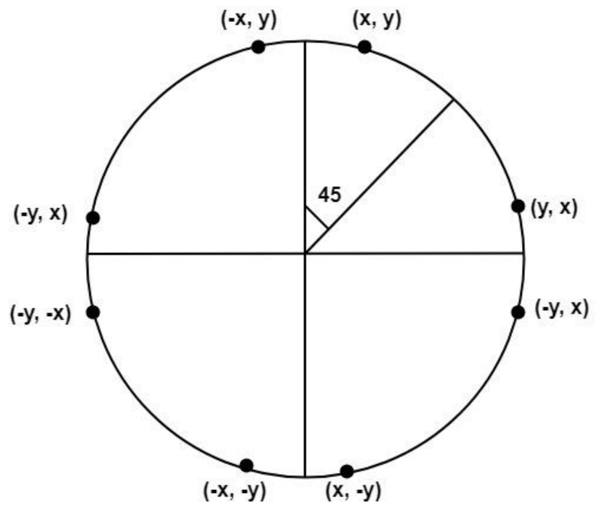
The Midpoint Line Algorithm is a foundational concept in computer graphics and has inspired various line-drawing algorithms and techniques. Its simplicity and efficiency make it a preferred choice for many line drawing applications, including rendering wireframes, drawing primitives, and implementing basic drawing functionalities in software graphics systems.

Defining a Circle:

Circle is an eight-way symmetric figure. The shape of circle is the same in all quadrants. In each quadrant, there are two octants. If the calculation of the point of one octant is done, then the other seven points can be calculated easily by using the concept of eight-way symmetry.

For drawing, circle considers it at the origin. If a point is $P_1(x, y)$, then the other seven points will be





So we will calculate only 45°arc. From which the whole circle can be determined easily.

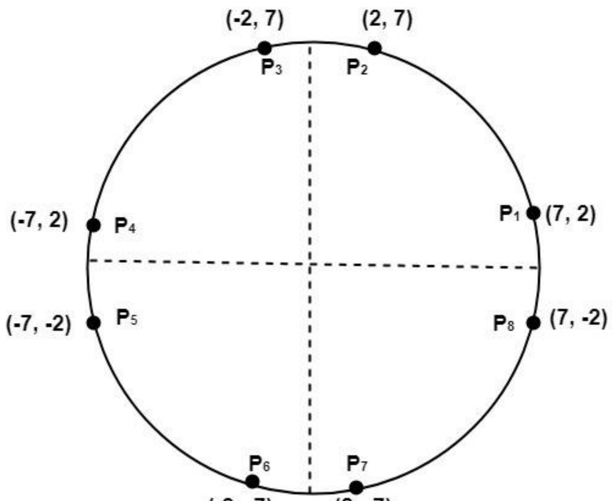
If we want to display circle on screen then the putpixel function is used for eight points as shown below:

putpixel (x, y, color)
putpixel (x, -y, color)
putpixel (-x, y, color)
putpixel (-x, -y, color)
putpixel (y, x, color)
putpixel (y, -x, color)
putpixel (-y, x, color)
putpixel (-y, -x, color)

Example: Let we determine a point (2, 7) of the circle then other points will be (2, -7), (-2, -7), (-2, 7), (7, 2), (-7, 2), (-7, -2), (7, -2)

These seven points are calculated by using the property of reflection. The reflection is accomplished in the following way:

The reflection is accomplished by reversing x, y co-ordinates.



Eight way symmetry of a Circle

There are two standards methods of mathematically defining a circle centered at the origin.

1. Defining a circle using Polynomial Method
2. Defining a circle using Polar Co-ordinates

1. Direct or Polynomial Method

In this method, a circle is defined with the help of a polynomial equation i.e.

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

Where, **(x_c, y_c)** are coordinates of circle and **r** is radius of circle.

For each value of **x**, value of **y** can be calculated using,

$$y = y_c \pm \sqrt{r^2 - (x - x_c)^2}$$

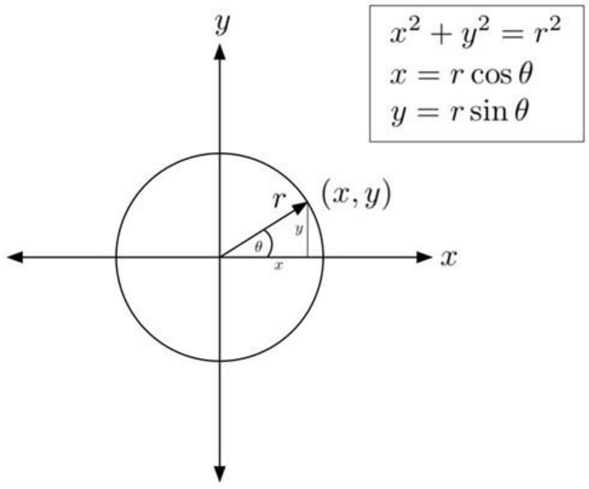
The initial points will be: **$x = x_c - r$**, and **$y = y_c$**

This is a very ineffective method because for each point value of **x_c**, **x** and **r** are squared and then subtracted and then the square root is calculated, which leads to high time complexity.

2. Polar coordinates Method

In this method, the coordinates are converted into polar coordinates. The coordinates are thus given by:

```
    x = r cosθ                  ...(1)
    y = r sinθ                  ...(2)
    r = radius
    Where, θ = current angle
```



For the first octant i.e. from θ = 0^0 to θ = 45^0,

The values of x and y are calculated using equations (1) and (2).

Simultaneously, all the eight points for the rest of the octants can be calculated using the eight-way symmetry property of the circle.

Then, the plotting of the circle is done.

Circle Generation Algorithm

Circle drawing in computer graphics is the process of approximating and rasterizing circles on a pixel grid. Since a circle is a continuous curve, representing it pixel by pixel requires specialized algorithms to achieve accurate and visually appealing results. There are several algorithms used for circle drawing in computer graphics. Here, we'll focus on the popular Midpoint Circle Algorithm (Bresenham's Circle Algorithm):

Midpoint Circle Algorithm (Bresenham's Circle Algorithm): The Midpoint Circle Algorithm is an efficient and widely used method for drawing circles in computer graphics. It uses the concept of symmetry to draw only one-eighth of the circle's circumference and then mirrors the points to complete the circle. The algorithm takes advantage of the fact that, for any given point on the circle's circumference, there are seven other symmetric points.

Steps of the Midpoint Circle Algorithm:

1. **Initialization:**
- Given the center of the circle (x_c, y_c) and its radius "r," set the initial point (x, y) as (0, r).
- Calculate the initial decision parameter "d" as: d = 1 - r.
2. **Pixel Plotting:**
- For each step in the algorithm, plot the current pixel (x, y) on the pixel grid, and set the corresponding symmetric points in the other octants.
3. **Incremental Approach:**
- At each step, evaluate the decision parameter "d" to determine whether the next pixel should be to the east (E) or southeast (SE) of the current pixel.
- Update the decision parameter "d" and the coordinates (x, y) based on the chosen pixel direction.
4. **Symmetry Handling:**
- The algorithm takes advantage of the symmetry properties of circles to draw only a specific octant (usually the first octant from 0° to 45°). For other octants, the algorithm mirrors the points by exchanging the roles of x and y and considering the corresponding octants.
5. **Drawing the Complete Circle:**
- Repeat the incremental steps until the pixel coordinates (x, y) move along the circle's circumference and cover all eight octants.
6. **Rasterization and Clipping:**

- Once the circle points are determined by the algorithm, they need to be rasterized onto the display screen, taking into account clipping to ensure that only visible parts of the circle are drawn.

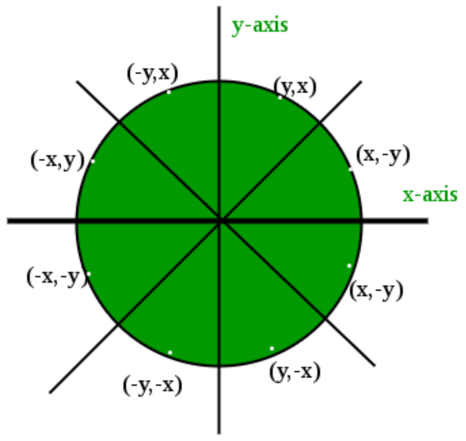
The Midpoint Circle Algorithm is highly efficient and produces circles with good visual quality. It utilizes only integer arithmetic, avoiding the need for costly floating-point calculations. The algorithm is commonly used in computer graphics applications for drawing circles, creating circular shapes, and rendering curves. Additionally, it serves as a foundation for more advanced circle and curve drawing techniques.

Bresenham’s circle drawing algorithm

It is not easy to display a continuous smooth arc on the computer screen as our computer screen is made of pixels organized in matrix form. So, to draw a circle on a computer screen we should always choose the nearest pixels from a printed pixel so as they could form an arc. There are two algorithm to do this:

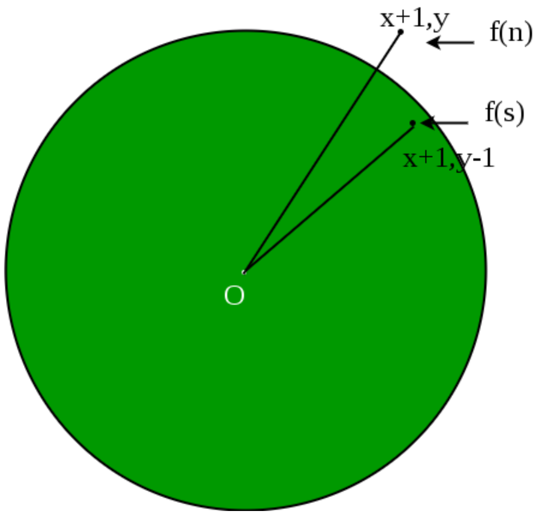
1. Mid-Point circle drawing algorithm
2. Bresenham’s circle drawing algorithm

We have already discussed the Mid-Point circle drawing algorithm in our previous post.In this post we will discuss about the Bresenham’s circle drawing algorithm. Both of these algorithms uses the key feature of circle that it is highly symmetric. So, for whole 360 degree of circle we will divide it in 8-parts each octant of 45 degree. In order to do that we will use Bresenham’s Circle Algorithm for calculation of the locations of the pixels in the first octant of 45 degrees. It assumes that the circle is centered on the origin. So for every pixel (x, y) it calculates, we draw a pixel in each of the 8 octants of the circle as shown below :



For a pixel (x,y) all possible pixels in 8 octants.

Now, we will see how to calculate the next pixel location from a previously known pixel location (x, y). In Bresenham's algorithm at any point (x, y) we have two option either to choose the next pixel in the east i.e. (x+1, y) or in the south east i.e. (x+1, y-1).



And this can be decided by using the decision parameter d as:

- If d > 0, then (x+1, y-1) is to be chosen as the next pixel as it will be closer to the arc.
- else (x+1, y) is to be chosen as next pixel.

Now to draw the circle for a given radius 'r' and centre (xc, yc) We will start from (0, r) and move in first quadrant till x=y (i.e. 45 degree). We should start from listed initial condition:

d = 3 - (2 * r)

x = 0

y = r