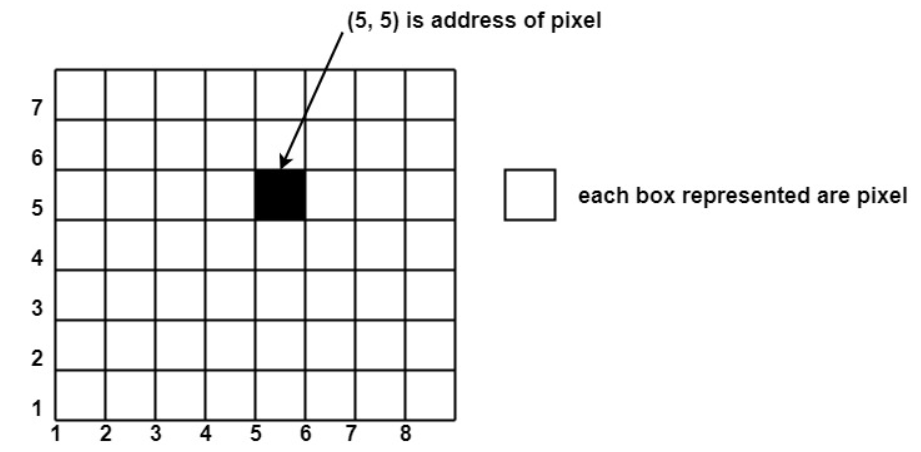
**Chapter 3**

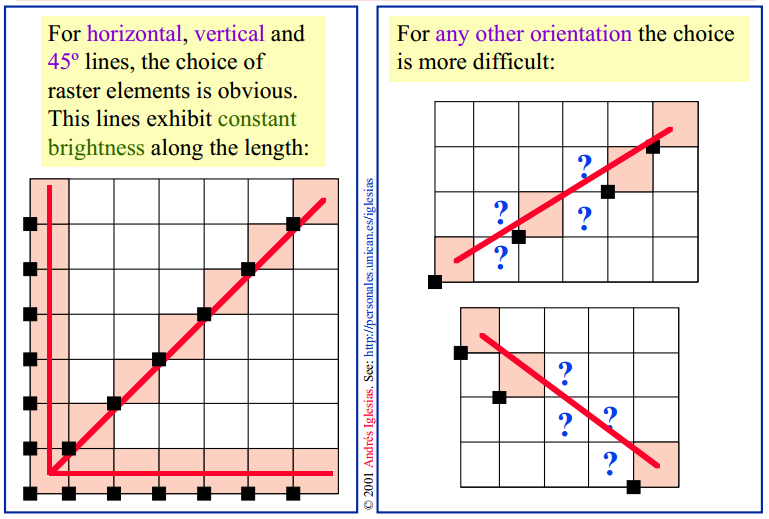
**Graphics primitives**

* **Scan Conversion Definition**
* It is a process of representing graphics objects as a collection of pixels. The graphics objects are continuous. The pixels used are discrete. Each pixel can have either on or off state.
* The circuitry of the video display device of the computer is capable of converting binary values (0, 1) into a pixel on and pixel off information. 0 is represented by pixel off. 1 is represented using pixel on. Using this ability, graphics computers represent pictures having discrete dots.
* Any model of graphics can be reproduced with a dense matrix of dots or points. Most human beings think of graphics objects as points, lines, circles, ellipses. For generating graphical objects, many algorithms have been developed.
* Many pictures, from *2D* drawings to projected *views* of 3D objects, consist of graphical primitives such as points, lines, circles, filled polygons, etc.
* These picture components are often defined in a continuous space at a higher level of abstraction than individual pixels in the discrete image space.
* A line is defined by its two endpoints and the line equation. A circle is defined by its radius, center position, and circle equation.
* **Advantage of developing algorithms for scan conversion**
* Algorithms can generate graphics objects at a faster rate.
* Using algorithms memory can be used efficiently.
* Algorithms can develop a higher level of graphical objects.
* **Examples of objects which can be scan converted**
* Point
* Line
* Sector
* Arc
* Ellipse
* Rectangle
* Polygon
* Characters
* Filled Regions
* **Pixel**
* The term pixel is a short form of the picture element. It is also called a point or dot. It is the smallest picture unit accepted by display devices. A picture is constructed from hundreds of such pixels. Pixels are generated using commands. Lines, circle, arcs, characters; curves are drawn with closely spaced pixels. To display the digit or letter matrix of pixels is used.
* The closer the dots or pixels are, the better will be the quality of picture. Picture will not appear jagged and unclear if pixels are closely spaced. So the quality of the picture is directly proportional to the density of pixels on the screen.
* Pixels are also defined as the smallest addressable unit or element of the screen. Each pixel can be assigned an address as shown in fig:



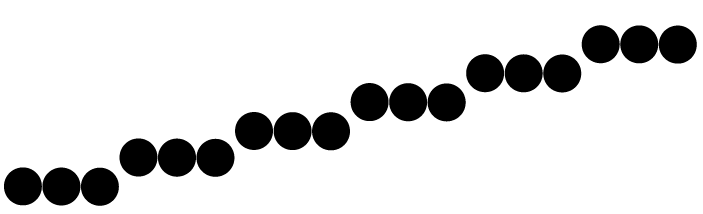
*Figure 1: Pixel representation*

* Different graphics objects can be generated by setting the different intensity of pixels and different colors of pixels. Each pixel has some co-ordinate value. The coordinate is represented using row and column.
* P (5, 5) used to represent a pixel in the 5th row and the 5th column. Each pixel has some intensity value which is represented in the memory of a computer called a **frame buffer**. Frame Buffer is also called a refresh buffer. This memory is a storage area for storing pixels’ values using which pictures are displayed. It is also called digital memory.
* Inside the buffer, the image is stored as a pattern of binary digits either 0 or 1. So there is an array of 0 or 1 used to represent the picture. In black and white monitors, black pixels are represented using 1's and white pixels are represented using 0's. In case of systems having one bit per pixel frame buffer is called a bitmap. In systems with multiple bits per pixel it is called a pixmap.
* **Rasterization:**
* Process of determining which pixels provide the best approximation to a desired line on the screen.
* **Scan conversion or rasterization** is the process of converting the primitives from its geometric definition into a set of pixels that make the primitive in image space.



*Figure 2: Rasterization*

Screen locations are referenced with integer values, so plotted positions may only approximate actual line positions between two specified endpoints. For example, line position of (12.36, 23.87) would be converted to pixel position (12, 24). This rounding of coordinate values to integers causes lines to be displayed with a stair step appearance (“the jaggies”), as represented in fig 3.



*Figure 3: The jaggies issue*

* **scan conversion:**
* Is how the line created with pixels, or how the line can know his way through 2 points.
* The process of representing continuous graphics objects as a collection of discrete pixels is called Scan Conversion.
* Example: a line is defined by its two end pts & the line equation, whereas a circle is defined by its radius, centre position & circle equation.
* Therefore, the process of making these coordinates according to the system's assumption i.e. (3,5) to plot the pixel is scan conversion.

**Points to be remember:**

* All the objects should be drawn with constant brightness.
* Objects should be independent of length & orientation.
* Scanning is done pixel by pixel, which means in the starting point an algorithm will have determined what is the next pixel and that will be repeated to get to the last point.

1. **Scan converting a Point**

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

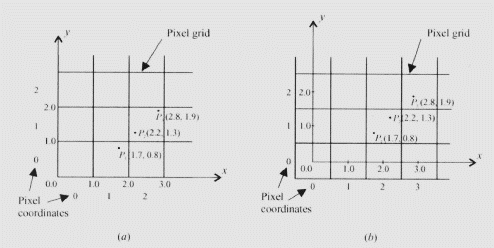
x’= Floor(x) and y’= Floor(y)

* ***Floor*** is a function that returns the largest integer that is less than or equal to the argument.
* Another approach is to scan convert (x, y) by making:

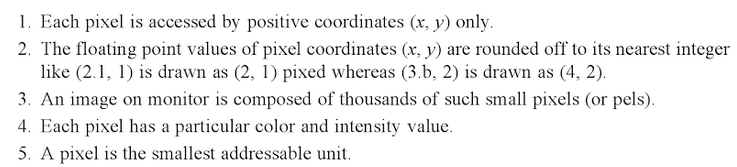
x’ = Floor (x + 0.5)

y’ = Floor(y+0.5).

* All points that satisfy x' <= x < x' + 1 and y' <= y < y' + 1 are mapped to pixels (x’, y').
* **A for example,** point P1 (1.7, 0.8) is represented by pixel (1,0). Point P2 (2.2, 1.3) and P3(2.8, 1.9) are both represented by pixel (2,1).
* Another approach is to align the integer values in the coordinate system for (x, y) by making x' = Floor(x+0.5) and y' = Floor(y+0.5).
* All points that satisfy x' - 0.5 <=x< x' + 0.5 and y' - 0.5 <=y< y' + 0.5 are mapped to pixel (x', y').



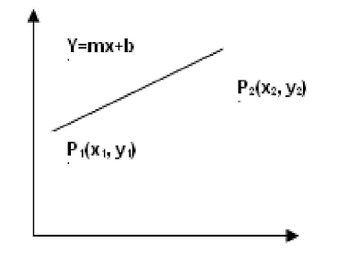
* Each pixel has certain characteristics:



1. **Line scan conversion:**

* Lines in computer graphics refer to a line segment which is a part of a straight line that extends infinitely in opposite directions.
* A line is defined by two ***end points*** and is represented by the equation **y=mx+b**

Where **m**: slope, **b**: the intercept of the line.

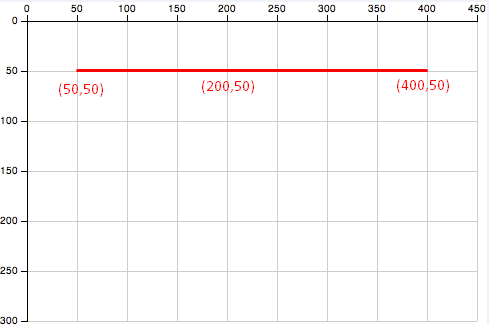


*Figure 4: Line equation*

* **Characteristics of good line:**

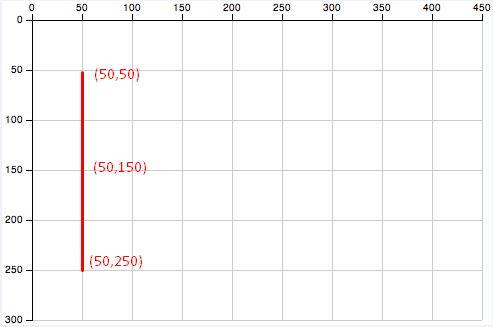
**Goals** (not all of them are achievable with the discrete space of a raster device):

* Straight lines should appear straight.
* Lines should start and end accurately, matching endpoints with connecting lines.
* Lines should have constant brightness.
* Lines should be drawn as rapidly as possible.
* **Horizontal Line:**
* Draw pixel P and increment x coordinate value by 1 to get the next pixel.



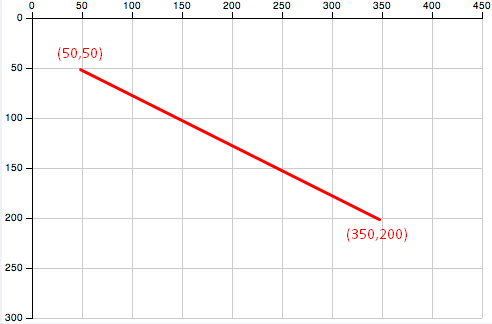
*Figure 5: Horizontal line representation*

* **Vertical Line:**
* Draw pixel P and increment y coordinate value by 1 to get the next pixel.



*Figure 6: Vertical line representation*

* **Diagonal Line:**
* Draw pixel P and increment both x and y coordinate by 1 to get the next pixel.

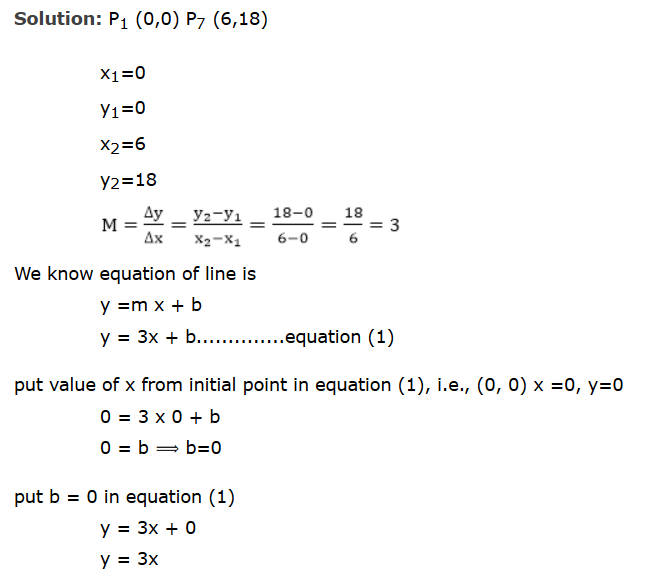


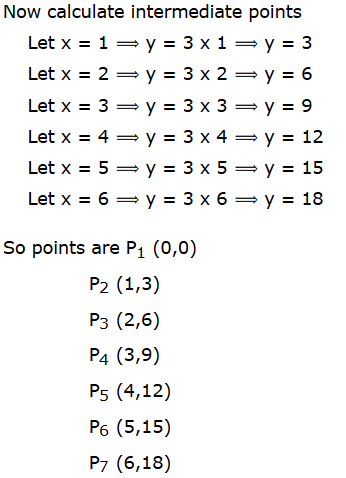
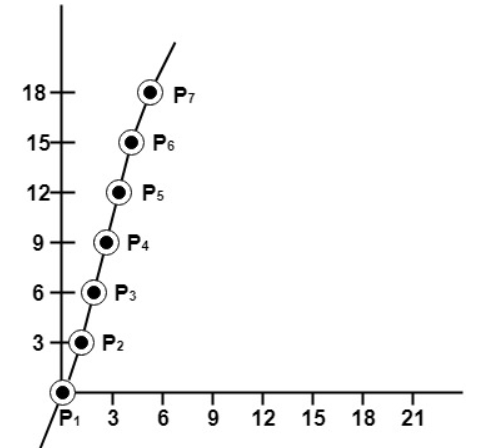
*Figure 7: Diagonal line representation*

* **Slope:**
* Slope helps us to determine if the X or the Y will increase in the next pixel.

Slope = m = dy/dx = (y2 - y1) / (x2 - x1)

* **To scan a line, we have two methods:**
* Direct use of the line equation
* DDA algorithm
* Bresenham's Algorithm
* **Direct use of the line equation**
* Scan-convert P1 and P2 to pixel coordinates (x1’, y1'), and (x2’, y2') respectively.
* Calculate m and b
  + Set m = (y2' -y1')/ (x2'-x1’) and b = y1' - mx1'.
* Check the value of m
  + If |m| < =1, for every integer value of x between and excluding x1' and x2', calculate the corresponding value of y using the equation and scan-convert (x, *y).*
  + If |m| >1, for every integer value of y between and excluding y1' and y2', calculate the corresponding value of x using the equation and scan-convert (x, *y).*
* **Example**: A line with starting point as (0, 0) and ending point (6, 18) is given. Calculate value of intermediate points and slope of line.



* **Line Raster using DDA (Digital Differential Analyser Algorithm)**
* DDA is an incremental scan conversion method.
* A vector generation algorithm that steps along the line to determine the pixels which should be turned-on
* DDA is simply a scan conversion line drawing algorithm based on the parameters
* **Algorithm:**
  + Scan convert P1 and P2 to pixel coordinates (x1', y1') and (x2', y2') respectively.
  + Calculate m = y2'-y1'/x2'-x1'
  + **if (|m| ≤ 1) then:** 
    - start with x=x1 and y=y1 and set
    - Compute successive y values as:

yi+1= yi + m

* + - * **y value is rounded off to the nearest integer to correspond to a screen pixel.**
  + **If (|m|>1), then:**
    - Start with x=x1 and y=y1 and set
    - Compute successive x values as:

X**i+1** = x**i** + 1/m

* This process continues until x reaches x2'(for the |m| <= 1 case) or y reaches y2' (for the |m| > 1 case) and all points found are scan-converted to pixel coordinates.
* **Example:**
* we want to draw a line from (0, 1) to (5,4)
* **Solution:**
* we first should calculate the slope

slope = (y2-y1)/(x2-x1) = (4-1)/ (5-0) = 3/5 so |m| <1

**Yi+1 = Yi + m**

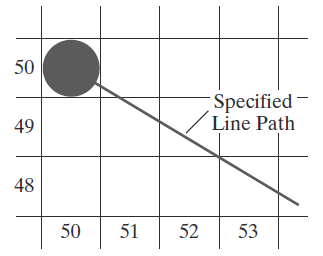
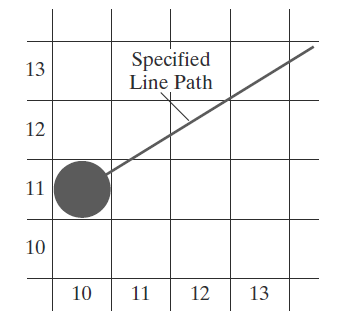
|  |  |  |  |
| --- | --- | --- | --- |
| **X** | yi+1=yi+m | **Round (yi)** | **yi** |
| 1 | 1+3/5 | Round(8/5) | 2 |
| 2 | 8/5+3/5 | Round(11/5) | 2 |
| 3 | 11/5+3/5 | Round(14/5) | 3 |
| 4 | 14/5+3/5 | Round(17/5) | 3 |
| 5 | 17/5+3/5 | Round(20/5) | 4 |

* **Advantages:**
* It is a faster algorithm.
* It is a simple algorithm.
* Require no special skills for implementation.
* **Disadvantages:**
* Floating point arithmetic is time consuming.
* Poor endpoint accuracy.

**Bresenham’s Line Algorithm**

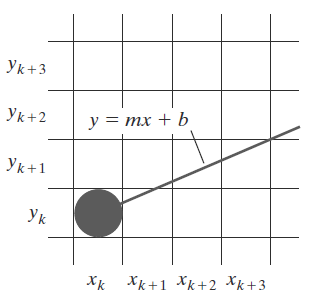
In this section, we introduce an accurate and efficient raster line-generating algorithm, developed by Bresenham, that uses only incremental integer calculations. In addition, Bresenham’s line algorithm can be adapted to display circles and other curves.

Figures below 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 the first figure, 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, the second figure 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 by testing the sign of an integer parameter whose value is proportional to the difference between the vertical separations of the two pixel positions from the actual line path

****

*Figure 8: determine the next position*

To illustrate Bresenham’s approach, we first consider the scan-conversion process for lines with ***positive slope less than 1.0***. Pixel positions along a line path are then determined by sampling at unit x intervals. Starting from the left endpoint (x0, y0) 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. Figure below demonstrates the kth step in this process. Assuming that we have determined that the pixel at (xk, yk) is to be displayed, we next need to decide which pixel to plot in column xk+1. Our choices are the pixels at positions (xk + 1, yk ) and (xk + 1, yk + 1).

****

*Figure 9:The K step in drawing the line*

***Let’s see Bresenham’s line drawing algorithm for |m| < 1***

1. Input the two line endpoints and store the left endpoint in (x0, y0).

2. Load (x0, y0) into the frame buffer; that is, plot the first point.

3. Calculate constants Δx, Δy, 2Δy, and 2Δy − 2Δx, and obtain the starting value for the decision parameter as

p0 = 2Δy − Δx

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

If pk < 0, the next point to plot is (xk + 1, yk) and

𝑝𝑘+1 = 𝑝𝑘 + 2Δ𝑦

Otherwise, the next point to plot is (xk + 1, yk + 1) and

𝑝𝑘+1 = 𝑝𝑘 + 2Δ𝑦 − 2Δ𝑥

5. Repeat step-4 Δx times.

For a line with positive slope greater than 1.0, we interchange the roles of the x and y directions. That is, we step along the y direction in unit steps and calculate successive x values nearest the line path. Also, we could revise the program to plot pixels starting from either endpoint. If the initial position for a line with positive slope is the right endpoint, both x and y decrease as we step from right to left.

**Example**

**Digitize the line with endpoints (20, 10) and (30, 18).**

This line has a slope of 0.8, with x = 10, y = 8

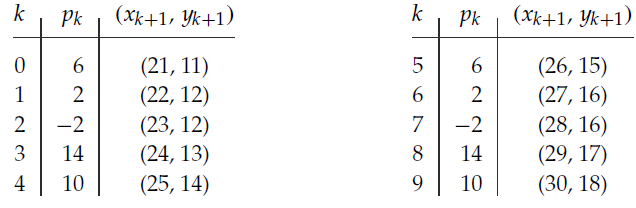
The initial decision parameter has the value

p0 = 2Δy −Δx=6

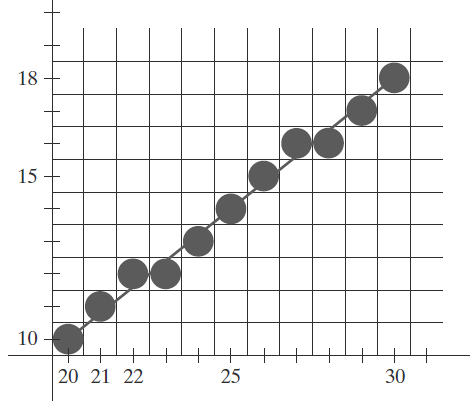
and the increments for calculating successive decision parameters are

2 Δy = 16, 2Δy − 2Δx = −4

We plot the initial point (x0, y0) = (20, 10), and determine successive pixel positions along the line path from the decision parameter as follows:



A plot of the pixels generated along this line path is shown



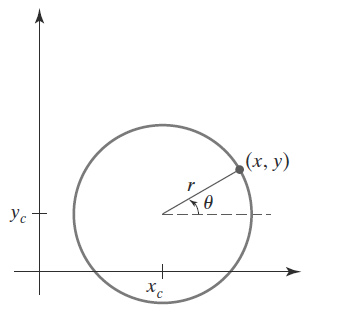
***The advantages of Bresenham Line Drawing Algorithm are-***

1. It is easy to implement.
2. It is fast and incremental.
3. It executes fast but less faster than DDA Algorithm.
4. The points generated by this algorithm are more accurate than the DDA Algorithm.
5. It uses fixed points only.

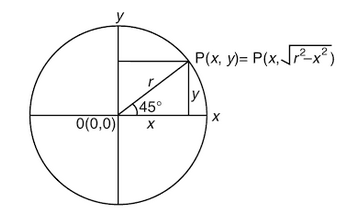
***The disadvantages of Bresenham Line Drawing Algorithm are-***

* Though it improves the accuracy of generated points but still the resulting line is not smooth.
* This algorithm is for the basic line drawing.
* It can not handle diminishing jaggies.

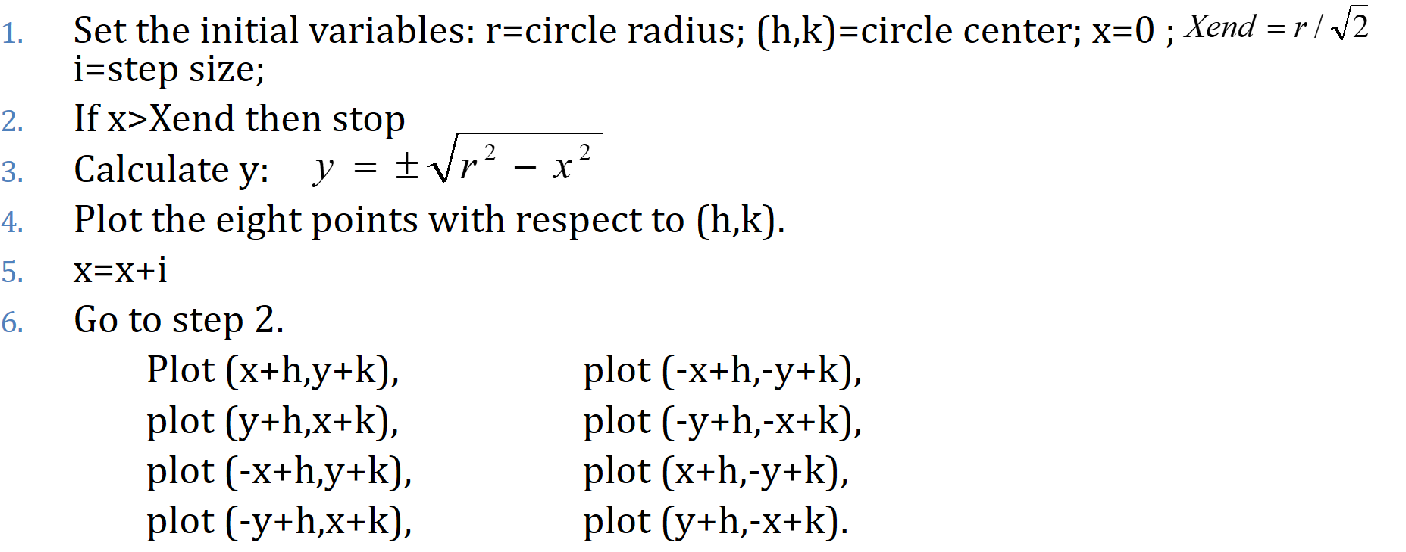
1. **Circle Raster:**



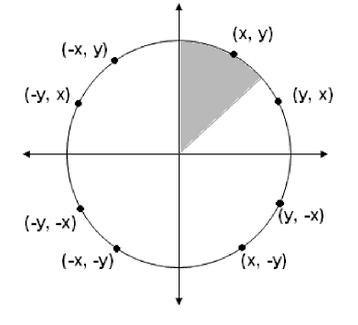
* **To raster a circle we need two things:**
* Centre coordinates.
* Radius
* **There are two standard methods of mathematically defining a circle centered at the origin.**
* Polynomial method
* Trigonometric method
* **Polynomial method:**
* Assume that there is a circle with its centre at origin (0, 0). Let the coordinates of a point on the circle and r be its radius



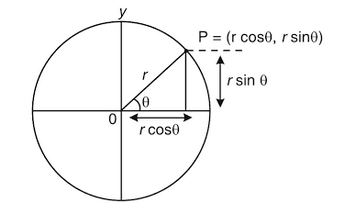
**Polynomial algorithm**



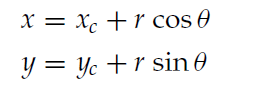
* **The circle reflections:**



* + **Advantages:**
    - Easy to understand
    - Fair to implement
  + **Disadvantages:** 
    - Consume time for calculations square root and square
* **Trigonometric method**



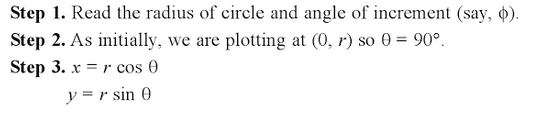
**From the shape:**

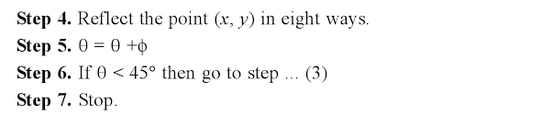


where (xc,yc) are the center position

**This method scan converts a circle by stepping from 0 to and then calculating each value of x and y.**

**Algorithm:**



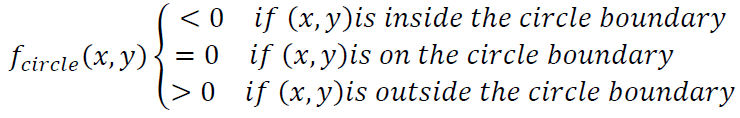


**Midpoint Circle Algorithm**

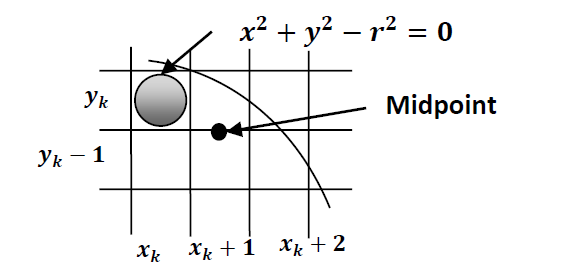
As in the raster line algorithm, we sample at unit intervals and determine the closest pixel position to the specified circle path at each step. For a given radius r and screen center position (xc , yc ), we can first set up our algorithm to calculate pixel positions around a circle path centered at the coordinate origin (0, 0). Then each calculated position (x, y) is moved to its proper screen position by adding xc to x and yc to y. Along the circle section from x = 0 to x = y in the first quadrant, the slope of the curve varies from 0 to −1.0. 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 pixel positions in any column is vertically closer to the circle path. Positions in the other seven octants are then obtained by symmetry. To apply the midpoint method, we define a circle function as



Any point (x, y) on the boundary of the circle with radius r satisfies the equation fcirc (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 as follows:



In the above equation, we calculate for the mid positions between pixels near the circular path at each sampling step and we set up incremental calculation for this function as we did in the line algorithm. Below figure shows the midpoint between the two candidate pixels at sampling position 𝑥𝑘 + 1.



Assuming we have just plotted the pixel at (𝑥𝑘, 𝑦𝑘) and next we need to determine whether the pixel at position (𝑥𝑘 + 1, 𝑦𝑘) or the one at positi) (𝑥𝑘 + 1, 𝑦𝑘 − 1) is closer to circle boundary.

**Algorithm for Midpoint Circle Generation**

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

(𝑥0, 𝑦0) = (0, 𝑟)

2. calculate the initial value of the decision parameter as

𝑝0 = 5/4 − 𝑟

3. At each xk position, starting at k = 0, perform the following test:

If px < 0, the next point along the circle centered on (0, 0) is (𝑥𝑘 + 1, 𝑦𝑘)&

𝑝𝑘+1 = 𝑝𝑘 + 2𝑥𝑘+1 + 1

Otherwise, the next point along the circle is (𝑥𝑘 + 1, 𝑦𝑘 − 1) &

𝑝𝑘+1 = 𝑝𝑘 + 2𝑥𝑘+1 + 1 − 2𝑦𝑘+1

                     Where 2𝑥𝑘+1 = 2𝑥𝑘 + 2, & 2𝑦𝑘+1 = 2𝑦𝑘 − 2

4. Determine symmetry points in the other seven octants.

5. Move each calculated pixel position (x, y) onto the circular path centered on (𝑥𝑐, 𝑦𝑐) and plot the coordinate values:

𝑥 = 𝑥 + 𝑥𝑐, 𝑦 = 𝑦 + 𝑦𝑐

6. Repeat steps 3 through 5 until 𝑥 ≥ 𝑦.

**Example**:

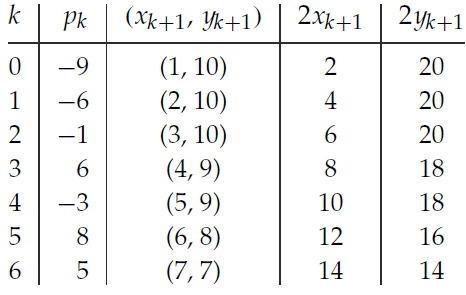
Given a circle radius r = 10, we demonstrate the midpoint circle algorithm by determining positions along the circle octant in the first quadrant from x = 0 to x = y. The initial value of the decision parameter is

p0 = 1 − r = −9

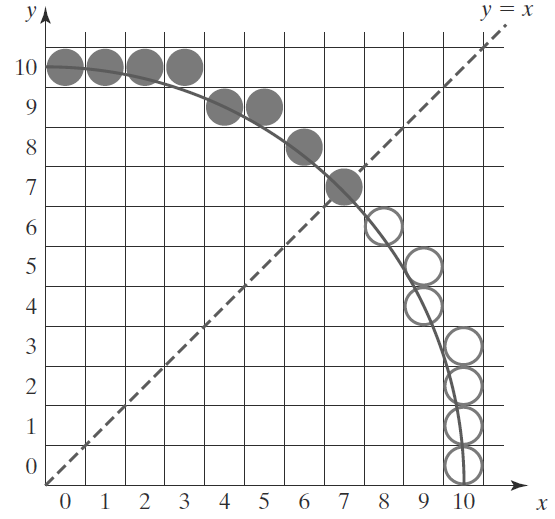
For the circle centered on the coordinate origin, the initial point is (x0, y0) = (0,10), and initial increment terms for calculating the decision parameters are

2x0 = 0, 2y0 = 20

Successive midpoint decision parameter values and the corresponding coordinate positions along the circle path are listed in the following table:

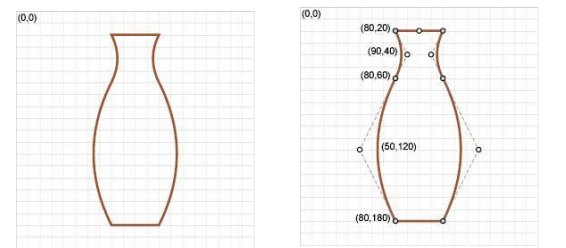


A plot of the generated pixel positions in the first quadrant is shown in Figure

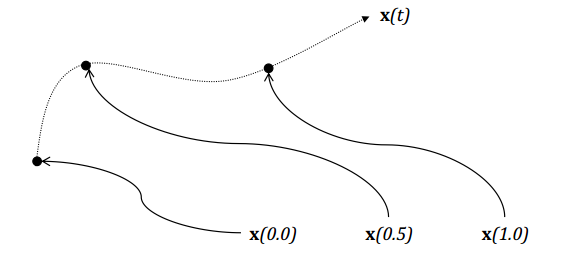


1. **Curves:**

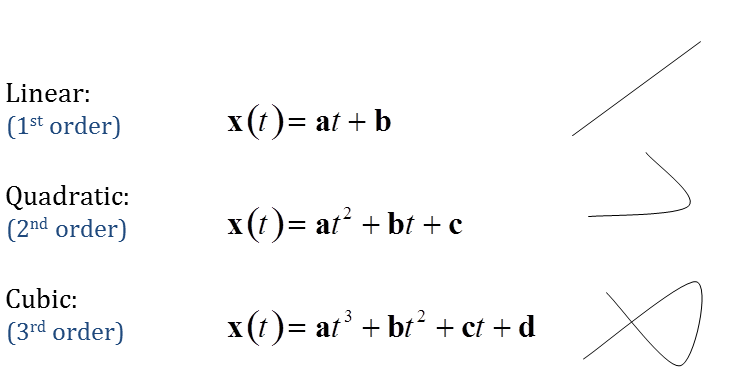
* Usefulness of curves in Computer Graphics:
  + Modelling different shapes.
  + Animation
    - Describe paths for the motion of objects in animations.
  + Fonts
    - Curves are widely used (Adobe, Microsoft) for font definition.
* **How to represent curves:**
* Specify every point along a curve.
* Used sometimes as “freehand drawing mode” in 2D applications.
  + Hard to get **exact results**.
  + Too much data, too hard to work with generally.
* Specify a curve using a small number of “control points”



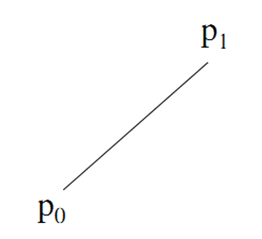
* **What is a curve, anyway?**
* Mathematically we treat it as a polynomial function, x(t)
* Given a value of t, computes a point x



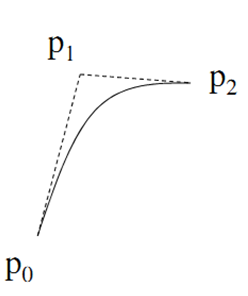
* **Types of Curve Curves**



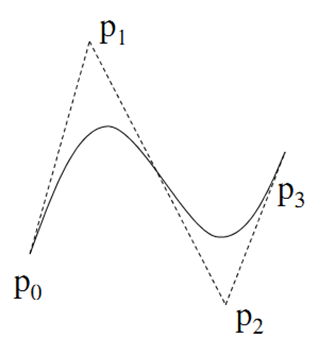
* We usually define the curve for 0 ≤ t ≤ 1
* Two points define a line (1st order):



* Three points define a quadratic curve (2nd order)



* Four points define a cubic curve (3rd order):



* + - * **k+1** points define a **k-**order curve