RGPVNOTES.IN

Program : **B.Tech**

Subject Name: **Computer Graphics & Multimedia**

Subject Code: **IT-601**

Semester: **6th**

Department of Information Technology

Subject Name: Computer Graphics and Multimedia **Subject Code:**IT-601

Subject Notes

Unit II

Scan conversion techniques, image representation, line drawing, simple DDA, Bresenham's Algorithm, Circle drawing, general method, symmetric DDA, Bresenham's Algorithm, curves, parametric function, Bezier Method, B-spline Method.

Scan conversion techniques:

The process of determining which pixels provide best approximation to the desired line is known as rasterization. When combined with the process of generating the picture in scan line order it is known as scan conversion.

Scan conversion involves changing the picture information data rate and wrapping the new picture in appropriate synchronization signals. There are two distinct methods for changing a picture's data rate:

a) Analog Methods (Non retentive, memory-less or real time method)

This conversion is done using large numbers of delay cells and is appropriate for analog video. It may also be performed using a specialized scan converter vacuum tube. In this case polar coordinates (angle and distance) data from a source such as a radar receiver, so that it can be displayed on a raster scan (TV type) display.

b) Digital methods (Retentive or buffered method)

In this method, a picture is stored in a line or frame buffer with n1 speed (data rate) and is read with n2 speed, several picture processing techniques are applicable when the picture is stored in buffer memory including kinds of interpolation from simple to smart high order comparisons, motion detection and … to improve the picture quality and prevent the conversion artefacts

.

Image representation:

Point is the fundamental element of the image representation. It is nothing but the position in a plane defined as either pairs or triplets of numbers depending on whether the data are two or three dimensional. Thus, (x1, Y1) or (x1, y1, z1) would represent a point in either two- or three-dimensional space. Two points would represent a line or edge, and a collection of three or more points a polygon. The representation of curved lines is usually accomplished by approximating them by short straight-line segments.

Line drawing

The process of 'turning on' the pixels for a line segment is called vector generation. In this section, we study the vector generation algorithm, Bresenham's line algorithm and circle drawing algorithms. Before discussing specific line drawing algorithms, it is useful to note the general requirements for such algorithms. These requirements specify the desired characteristics of line.

Characteristics of a line:

- The line should be appearing as a straight line and it should start and end accurately.
- The line should be displayed with constant brightness along its length independent of its length and orientation.
- The line should be drawn rapidly.

The basic concept behind all drawing algorithms is to reduce the computations and provide the result rapidly, so most of the line drawing algorithms use incremental methods. In these methods line starts with starting point and then a fix increment is added to get the next point on the line. This is continued till the end of line.

Simple DDA (Digital Differential Analyzer) Algorithm:

The Digital Differential Analyzer (DDA) is a Scan-Conversion line algorithm-based calculating either Δy or Δx.

Consider first a line with positive slope, less than or equal to 1, we sample at unit intervals (Δx=1) and compute each successive y value as $y_{k+1} = y_k + m$, subscript k takes integer values starting from 1, for the first point, and increases by 1 until the final endpoints is reached. Since m can be any real number between 0 & 1, the calculated y values must be rounded to the nearest integer. For lines with a positive slope greater than 1, we reserve the roles of x & y. That is, we sample at unit y intervals (Δy=1) and calculate each succeeding x value as $x_{k+1} = x_k + (1/m)$.

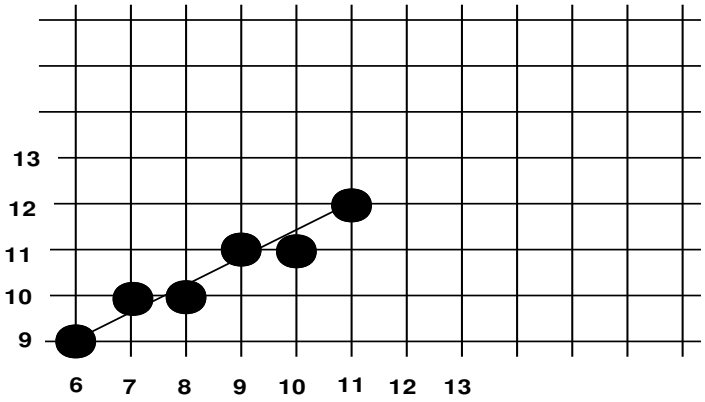

Figure 2.1(a) Representation of line with DDA algorithm

```
Procedure DDA (X1, Y1, X2, Y2: Integer);
Var     Length, I: Integer;
        X,Y,Xinc,Yinc:Real;
        Begin
        Length: = ABS(X2 - X1);
        If ABS (Y2 - Y1) > Length Then
                Length: = ABS (Y2-Y1);
        Xinc: = (X2 - X1)/Length;
        Yinc: = (Y2 - Y1)/Length;
        X: = X1+0.5*SIGN(Xinc);
        Y: = Y1+0.5*SIGN(Yinc);
For I: = 0 To Length Do
        Begin
                Plot (Round(X), Round(Y));
                X: = X + Xinc;
                Y: = Y + Yinc
        End {For}
End; {DDA}
```

DDA (*digital differential analyzer*) creates good lines but it is too time consuming due to the round function and long operations on real values.

Key points:

- DDA works with floating point arithmetic
- Rounding to integers necessary
- It takes lot of computation time because at each and every step it has to round off.

Advantages of DDA Algorithm

1. It is the simplest algorithm

2. It is a is a faster method for calculating pixel positions

Disadvantages of DDA Algorithm

1. Floating point arithmetic in DDA algorithm is still time-consuming
2. End point accuracy is poor

Symmetrical DDA:

The Digital Differential Analyzer (DDA) generates lines from their differential equations. The equation of a straight **line is**

$$\frac{dy}{dx} = \frac{\Delta y}{\Delta x}$$

The DDA works on the principle that we simultaneously increment x and y by small steps proportional to the first derivatives of x and y. In this case of a straight line, the first derivatives are constant and are proportional to Δx and Δy. Therefore, we could generate a line by incrementing x and y by ϵ Δx and ϵ Δy, where ϵ is some small quantity. There are two ways to generate points

1. By rounding to the nearest integer after each incremental step, after rounding we display dots at the resultant x and y.

2. An alternative to rounding the use of arithmetic overflow: x and y are kept in registers that have two parts, integer and fractional. The incrementing values, which are both less than unity, are repeatedly added to the fractional parts and whenever the results overflows, the corresponding integer part is incremented. The integer parts of the x and y registers are used in plotting the line. In the case of the symmetrical DDA, we choose $\varepsilon=2^{-n}$, where $2^{n-1} \leq \max(|\Delta x|, |\Delta y|) < 2^{n}$

A line drawn with the symmetrical DDA is shown in fig 2.1(b):

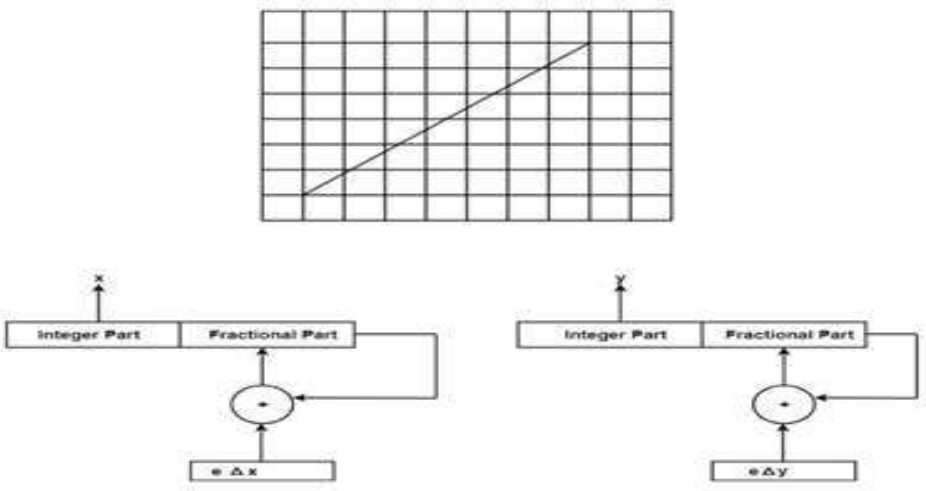

Figure 2.1(b)Symmetric DDA

Example: If a line is drawn from (0, 0) to (10, 5) with a symmetrical DDA

1. How many iterations are performed?
2. How many different points will be generated?

Solutions: Given: P^1 (0,0) P^2 (10,5)

$x_1=0$
$y_1=0$
$x_2=10$
$y_2=5$
dx = 10 - 0 = 10
dy = 5 - 0 = 0

$$m = \frac{y_2-y_1}{x_2-x_1} = \frac{5-0}{10-0} = \frac{5}{10} = 0.5$$

P_1 (0,0) will be considered starting points
P_3 (1,0.5) point not plotted
P_4 (2, 1) point plotted
P_5 (3, 1.5) point not plotted
P_6 (4,2) point plotted
P_7 (5,2.5) point not plotted
P_8 (6,3) point plotted
P_9 (7,3.5) point not plotted
P_{10} (8, 4) point plotted
P_{11} (9,4.5) point not plotted
P_{12} (10,5) point plotted

Following Figure show line plotted using these points.

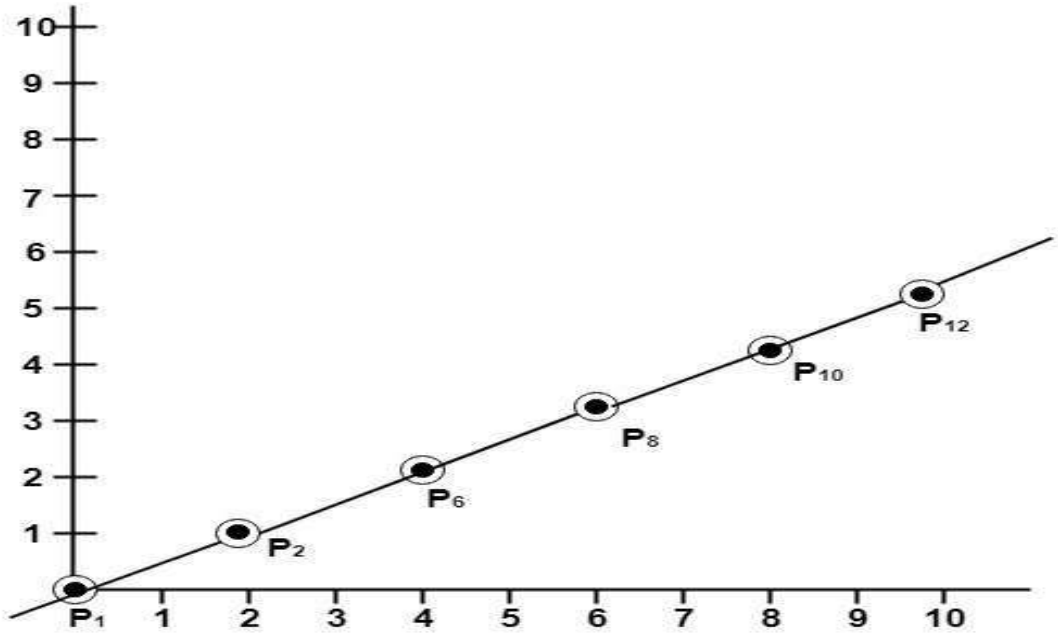

Figure 2.1(c) Line using Symmetric DDA

Bresenham's Line Algorithm:

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

We first consider the scan-conversion process for lines with positive slope less than 1. Pixel positions along a line path are then determined by sampling at unit x intervals. Starting from the left endpoint (x0, y0) of a line, we step to each successive column (x position) and plot the pixel whose scan-line y value is closest to the line path.

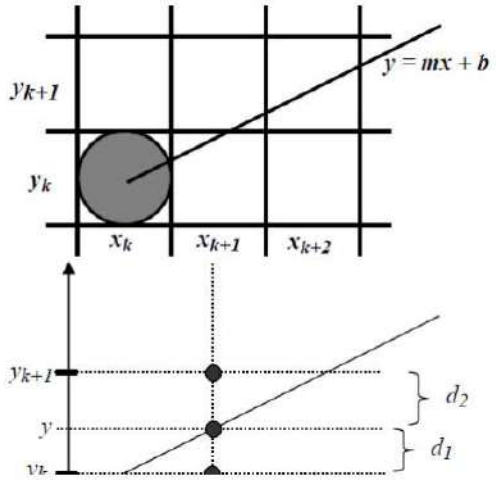

Figure 2.2 Concept of Bresenham's Line Drawing

The above figure demonstrates the kth step in this process. Assuming we have determined that the pixel at (x_k, y_k) is to be displayed, we next need to decide which pixel to plot in column x_{k+1}. Our choices are the pixels at positions (x_{k+1}, y_k) and (x_{k+1}, y_{k+1}). At sampling position x_{k+1}, we label vertical pixel separations from the mathematical line path as d_1 and d_2 shown in the diagram. The y coordinate on the mathematical line at pixel column position x_{k+1} is calculated as

Step 1. Input Two endpoints, store left endpoint in (x0, y0).

Step 2. Load (x0, y0) into frame buffer, i.e. plot the first point.

Step 3. Calculate constants Δx, Δy, 2Δy, 2Δy – 2 Δx, and initial value of decision parameter:
$P_0 = 2\Delta y - \Delta x$

Step 4. At each x_k along the line, start at k=0, test: if $P_k < 0$, plot (x_{k+1}, y_k) and $P_{k+1} = P_k + 2\Delta y$
ELSE
Plot (x_{k+1}, y_{k+1}) and $P_{k+1} = P_k + 2\Delta y - 2\Delta x$

Step 5. Repeat step (4) Δx times.

A) Bresenham's line algorithm for m<1

FOR (X1, Y1) =1, 2 AND (X2, Y2) =8, 5

K	PK	XK+1	YK+1
0	-1	2	2
1	5	3	3
2	-3	4	3
3	3	5	4
4	-5	6	4
5	1	7	5
6	-7	8	5

input line endpoints, (x_0,y_0) and (x_n, y_n)
calculate $\Delta x = x_n - x_0$ and $\Delta y = y_n - y_0$
calculate parameter $p_0 = 2\Delta y - \Delta x$
set pixel at position (x_0,y_0)

repeat the following steps until (x_n, y_n) is reached:

if $p_i < 0$

set the next pixel at position $(x_i + 1, y_i)$

calculate new $p_{i+1} = p_i + 2\Delta y$

if $p_i \geq 0$

set the next pixel at position $(x_i + 1, y_i + 1)$

calculate new $p_{i+1} = p_i + 2\Delta y - 2\Delta x$

Repeat steps until dx-1 times.

B) Bresenham's line algorithm for m>1

input line endpoints, (x_0, y_0) and (x_n, y_n)

calculate $\Delta x = x_n - x_0$ and $\Delta y = y_n - y_0$

calculate parameter $p_0 = 2\Delta x - \Delta y$

set pixel at position (x_0, y_0)

repeat the following steps until (x_n, y_n) is reached:

if $p_i < 0$

set the next pixel at position $(x_i, y_i + 1)$

calculate new $p_{i+1} = p_i + 2\Delta x$

if $p_i \geq 0$

set the next pixel at position $(x_i + 1, y_i + 1)$

calculate new $p_{i+1} = p_i + 2\Delta x - 2\Delta y$

Repeat steps until dy-1 times.

F0R (X1, Y1) =1, 3 AND (X2, Y2) =8, 12

K	PK	XK+1	YK+1
0	5	2	4
1	1	3	5
2	-3	3	6
3	11	4	7
4	7	5	8
5	3	6	9
6	-1	6	10
7	13	7	11
8	9	8	12

Key points

- Bresenham's algorithm uses integer arithmetic
- Constants need to be computed only once
- Bresenham's algorithm generally faster than DDA

Circle Drawing:

In this section we are going to discuss three efficient circle drawing algorithms:

1. General Method
2. Symmetric DDA algorithm
3. Bresenham's circle algorithm

1. General Method:

Using circle equation:

We could solve for y in terms of x

,

and use this equation to compute the pixels of the circle.

C program to rasterize the circle:-

```
public void circleSimple(int xCenter, int yCenter, int radius, Color c)
{
   int x, y, r2;
   r2 = radius * radius;
   for (x = -radius; x <= radius; x++) {
      y = (int) (Math.sqrt(r2 - x*x) + 0.5);
putpixel(xCenter + x, yCenter + y, WHITE);
putpixel(xCenter + x, yCenter – y, WHITE);
   }
}
```

Using Polar Co-ordinates Method:

Defining a circle using Polar Co-ordinates:

The second method of defining a circle makes use of polar coordinates as shown in fig 2.3:

$x = r \cos \theta$

$y = r \sin \theta$

Where θ=current angle

r = circle radius

x = x coordinate

y = y coordinate

By this method, θ is stepped from 0 to π/4 & each value of x & y is calculated.

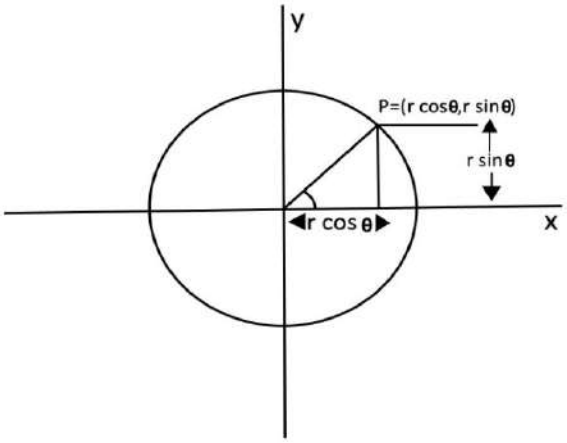

Figure 2.3 Polar Co-ordinates Method

Algorithm:

Step1: Set the initial variables:

r = circle radius

(h, k) = coordinates of the circle center

i = step size

θ_end=(22/7)/4

θ=0

Step2: If $\theta > \theta_{end}$ then stop.

Step3: Compute

x = r * cos θ y=r*sin?θ

Step4: Plot the eight points, found by symmetry i.e., the center (h, k), at the current (x, y) coordinates.

Plot (x + h, y +k) Plot (-x + h, -y + k)

Plot (y + h, x + k) Plot (-y + h, -x + k)

Plot (-y + h, x + k) Plot (y + h, -x + k)

Plot (-x + h, y + k) Plot (x + h, -y + k)

Step5: Increment θ=θ+i

Step6: Go to step (ii).

2. DDA Circle Drawing Algorithm We know that, the equation of circle, with origin as the center of the circle is given as $x^2 + y^2 = r^2$ The digital differential analyser algorithm can be used to draw the circle by defining circle as a differential equation. It is as given below,

Algorithm

1. Read the radius (R), of the circle and calculate value of epsilon
2. start_x = 0

   start_y = R
3. X1 = start_x

   Y1= start_y
4. do {

   X2=X1 + epsilon*Y1

   Y2 = Y1 + epsilon*X2

   Plot (int (X2), int (Y2))

   X1 = X2 ;

   Y1 = Y2 ;

} while (Y1— start_y) < epsilon OR (start_x — X1) > epsilon)

[check if the current point is the starting point or not if current point is not starting point repeat step 4 ; otherwise stop]

5. Stop.

3. Bresenham's Circle Algorithm:

Bresenham's circle drawing algorithm is used to determine the next pixel of screen to be illuminated while drawing a circle by determining the closest nearby pixel.

Let us first take a look how a circle is drawn on a pixel screen

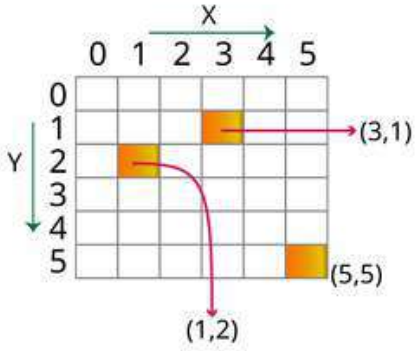

Figure2.4: pixel graph

As Circles are symmetrical so the values of y-intercept and x-intercept are same if circle's Center coordinates are at Origin (0,0).

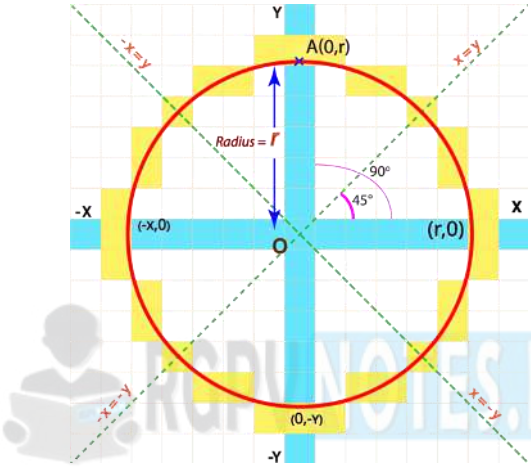

Figure 2.5 Circle Geometry

Here,

Radius = OA = r

Due to symmetrical property of Circle we don't need to calculate all the pixels of all the octets and quadrants. We need to find the pixels of only one octet, rest we can conclude through this.

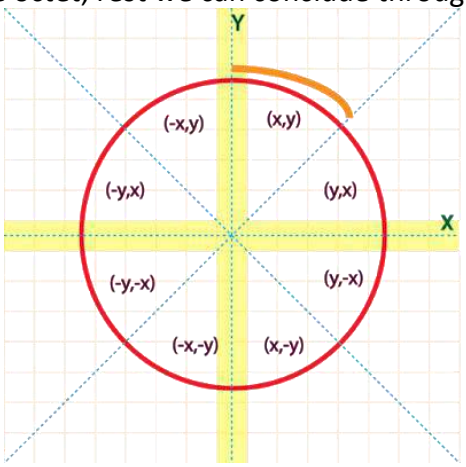

Figure 2.6 Eight-way Symmetry of circle

Lets take the Octet 2 which is in quadrant 1, here both x and y are positive and here the initial pixel would be (0,y) coordinate.

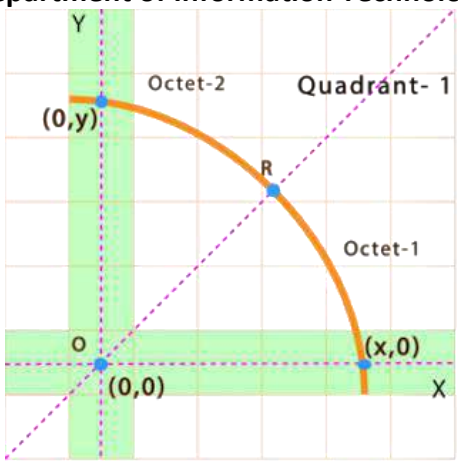

Figure 2.7: Octet 2

At point R both the value of both x and y coordinates would be same as R is at same distance of Both X and Y axis.

Derivation of Bresenham's circle algorithm:

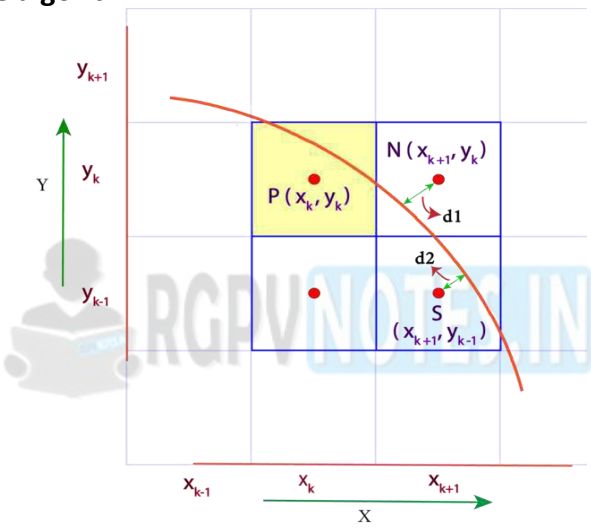

Figure 2.8: Bresenham's circle algorithm concept

Let's say our circle is at some random pixel P whose coordinates are (x_k, y_k).

Now we need to find out our next pixel.

Note- This is octet 2 so here x can never be decremented as per properties of a circle but y either needs to decremented or to be kept same. y is needed to be decided.Here it needs to decide whether go with N or S.

For this Bresenham's circle drawing algorithm will help us to decide by calculating the difference between radius and the coordinates of the next pixels.

The shortest of d1 and d2 will help us Decide our next pixel.

note- $x_{k+1} = x_k + 1$

As x_{k+1} is the next consecutive pixel of x_k

similarly

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

Equation of Circle with Radius r

$(x - h)^2 + (y - k)^2 = r^2$

When coordinates of Centre are at Origin i.e., (h=0, k=0)

x2 + y2 = r2 (Pythagoras theorem)

Function of Circle Equation

F(C) = x2 + y2 - r2

Function of Circle at N

F(N) = (xk+1)2 + (yk)2 – r2 (Positive)

Here the value of F(N) will be positive because N is out-side the circle that makes (xk+1)2 + (yk)2 Greater than r2

Function of Circle at S

F(S) = (xk+1)2 + (yk-1)2 – r2 (Negative)

Here the value of F(S) will be Negative because S is in-side the circle that makes (xk+1)2 + (yk-1)2 Less than r2

Now we need a decision parameter which help us decide the next pixel

Say D_k

And, $D_k = F(N)+F(S)$

Here either we will get the positive or negative value of D_k

So, if $D_k < 0$ that means the negative F(S) is bigger than the positive F(N), that implies Point N is closer to the circle than point S. So we will select pixel N as our next pixel.

and if $D_k > 0$ that means positive F(N) is bigger and S is closer as F(S) is smaller. So, we will Select S as our next pixel.

Now let'sfind Dk

$$D_k = (x_{k+1})^2 + (y_k)^2 - r^2 + (x_{k+1})^2 + (y_{k-1})^2 - r^2$$

(replacing x_{k+1} with $x_k + 1$ and y_{k-1} with $y_k - 1$)

$$= (x_k + 1)^2 + (y_k)^2 - r^2 + (x_k + 1)^2 + (y_k - 1)^2 - r^2$$

$$= 2(x_k + 1)^2 + (y_k)^2 + (y_k - 1)^2 - 2r^2 \quad \text{----- } (i)$$

Now let's find D_{k+1}

(Replacing every k with k+1)

$$D_{k+1} = 2(x_{k+1} + 1)^2 + (y_{k+1})^2 + (y_{k+1} - 1)^2 - 2r^2$$

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

(Replacing x_{k+1} with $x_k + 1$ but now we can't replace y_{k+1} because we don't know the exact value of y_k)

$$= 2(x_k + 1 + 1)^2 + (y_{k+1})^2 + (y_{k+1} - 1)^2 - 2r^2$$

$$= 2(x_k + 2)^2 + (y_{k+1})^2 + (y_{k+1} - 1)^2 - 2r^2 \quad \text{----- } (ii)$$

Now to find out the decision parameter of next pixel i.e. D_{k+1}

We need to find

$$D_{k+1} - D_k = (ii) - (i)$$

$$= 2(x_k + 2)^2 + (y_{k+1})^2 + (y_{k+1} - 1)^2 - 2r^2 - [2(x_k + 1)^2 + (y_k)^2 + (y_k - 1)^2 - 2r^2]$$

$$= 2(x_k)^2 + 8x_k + 8 + (y_{k+1})^2 + (y_{k+1})^2 - 2y_{k+1} + 1 - 2r^2 - 2x_k - 4x_k - 2 - (y_k)^2 - (y_k)^2 + 2y_k - 1 + 2r^2$$

$$= 4x_k + 2(y_{k+1})^2 - 2y_{k+1} - 2(y_k)^2 - 2y_k + 6$$

$$D_{k+1} = D_k + 4x_k + 2(y_{k+1})^2 - 2y_{k+1} - 2(y_k)^2 - 2y_k + 6 \quad \text{----- } (iii)$$

If ($D_k < 0$) then we will choose N point as discussed.i.e. (x_{k+1}, y_k)that means our next x coordinate is x_{k+1} and next y coordinate is y_k i.e. $y_{k+1} = y_k$, putting y_k in *(iii)* in replace of y_{k+1}

now,

$$D_{k+1} = D_k + 4x_k + 2(y_k)^2 - 2y_k - 2(y_k)^2 - 2y_k + 6$$

$$= D_k + 4x_k + 6$$

If ($D_k > 0$) then we will choose S point.

i.e. (x_{k+1}, y_{k-1})that means our next x coordinate is x_{k+1} and next y coordinate is y_k i.e. $y_{k+1} = y_{k-1}$ putting y_{k-1} in *(iii)* in replace of y_{k+1}

now,

$$D_{k+1} = D_k + 4x_k + 2(y_{k-1})^2 - 2y_{k-1} - 2(y_k)^2 - 2y_k + 6$$

Now we know

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

therefore,

$D_{k+1} = D_k + 4x_k + 2(y_k - 1)^2 - 2(y_k - 1) - 2(y_k)^2 - 2y_k + 6$

$= D_k + 4x_k + 2(y_k)^2 + 2 - 4y_k - 2y_k + 2 - 2(y_k)^2 - 2y_k + 6$

$= D_k + 4x_k - 4y_k + 10$

$= D_k + 4(x_k - y_k) + 10$

Now to find initial decision parameter means starting point that is (0,r) ,value of y is r

Putting (0,r) in (*i*)

$D_k = 2(x_k + 1)^2 + (y_k)^2 + (y_k - 1)^2 - 2r^2$

$D_0 = 2(0 + 1)^2 + r^2 + (r - 1)^2 - 2r^2$

$= 2 + r^2 + r^2 + 1 - 2r - 2r^2$

$= 3 - 2r$

Hence, we have derived the Bresenham's Circle drawing technique.

Algorithm pseudocode:

$x_0 = 0.$

$y_0 = r$

$p_0 = 3 - 2r$

if $p_i < 0$ *then*

$y_{i+1} = y_i$

$p_{i+1} = p_i + 4x_i + 6$

else if $p_i \geq 0$ *then*

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

$p_{i+1} = p_i + 4(x_i - y_i) + 10$

Stop when $x_i \geq y_i$ and determine symmetry points in the other octants

For radius=6

So your first point is (0, r) = (0, 6)

K	PI	XK+1	YK+1
0	-9	1	6
1	1	2	5
2	-1	3	5
3	17	4	4

Bresenham's Circle Algorithm:

For radius=6 suppose center is(xc,yc)=3

K	PI	XK+1	YK+1	PLOTTED POINTS
0	-9	1	6	X=xc+x y=yc+y X=3+1=4,y=3+6=9
1	1	2	5	X=xc+x y=yc+y X=3+2=5,y=3+5=8
2	-1	3	5	X=xc+x y=yc+y X=3+3=6,y=3+5=8
3	17	4	4	X=xc+x y=yc+y X=3+4=7,y=3+4=7

Curves:

In mathematics, a curve is an object similar to a line which does not have to be straight.

Parametric function:

Parameters for curve attributes are the same as those for line segments. We can display curves with varying colours, widths, dot dash patterns, and available pen or brush options. Raster curves of various widths can be displayed using the method of horizontal or vertical pixel spans. Where the magnitude of the curve slope is less than 1, we plot vertical spans; where the slope magnitude is greater than 1, we plot horizontal spans. Another method for displaying thick curves is to fill in the area between two parallel curve paths, whose separation distance is equal to the desired width.

We could do this using the specified curve path as one boundary and setting up the second boundary either inside or outside the original curve path. This approach shifts the original curve path either inward or outward, depending on which direction we choose for the second boundary. We can maintain the original curve position by setting the two boundary curves at a distance of one-half the width on either side of the specified curve path. Although this method is accurate for generating thick circles, it provides only an approximation to the true area of other thick curves. Curves drawn with pen and brush shapes can be displayed in different sizes and with superimposed patterns or simulated brush strokes.

Parametric equations can be used to generate curves that are more general than explicit equations of the form y=f(x). A quadratic parametric spline may be written as

$$P=a_2t^2+a_1t+a_0$$

where P is a point on the curve, a_0, a_1 and a_2 are three vectors defining the curve and t is the parameter. The curve passes through three points labelled P_0, P_1 and P_2. By convention the curve starts from point P_0 with parameter value t=0, goes through point P_1 when t=t1 (0<t1<1) and finishes at P2 when t=1. Using these conventions, we can solve for the three a vector as follows:

$$\begin{aligned} t = 0 \quad & \mathbf{P}_0 = \mathbf{a}_0 \\ t = 1 \quad & \mathbf{P}_2 = \mathbf{a}_2 + \mathbf{a}_1 + \mathbf{a}_0 \\ t = t_1 \quad & \mathbf{P}_1 = \mathbf{a}_2 t_1^2 + \mathbf{a}_1 t_1 + \mathbf{a}_0 \end{aligned}$$

and rearranging these equations we get:

$$\begin{aligned} \mathbf{a}_0 &= \mathbf{P}_0 \\ \mathbf{a}_2 &= \frac{(\mathbf{P}_1 - \mathbf{P}_0) - t_1(\mathbf{P}_2 - \mathbf{P}_0)}{t_1(t_1 - 1)} \\ \mathbf{a}_1 &= \mathbf{P}_2 - \mathbf{P}_0 - \mathbf{a}_2 \end{aligned}$$

Bezier Method:

Bezier curve is discovered by the French engineer **Pierre Bezier**. These curves can be generated under the control of other points. Approximate tangents by using control points are used to generate curve. The Bezier curve can be represented mathematically as –

Suppose we are given n+1 control point position:Pk = (Xk, Yk,Zk), with k = o to n. These coordinate points are blended to produce the position vector P(u), which describes the path of an approximating Bezier Polynomial Function between Po and Pn.

$$P(u) = \sum_{k=0}^{n} \mathrm{Pk\ BEZk,n(u)}\ 0<=u<=1$$

The Bezier blending functions $BEZ_{k,n}(u)$ are the Bernstein polynomials.

$$BEZ_{k,n}(u) = C(n,k)u^k(1-u)^{n-k}$$

Where,

C(n,k) are the binomial coefficients.

C(n,k) = n! / k!(n-k)!

The simplest Bézier curve is the straight line from the point P0P0 to P1P1. A quadratic Bezier curve is determined by three control points. A cubic Bezier curve is determined by four control points.

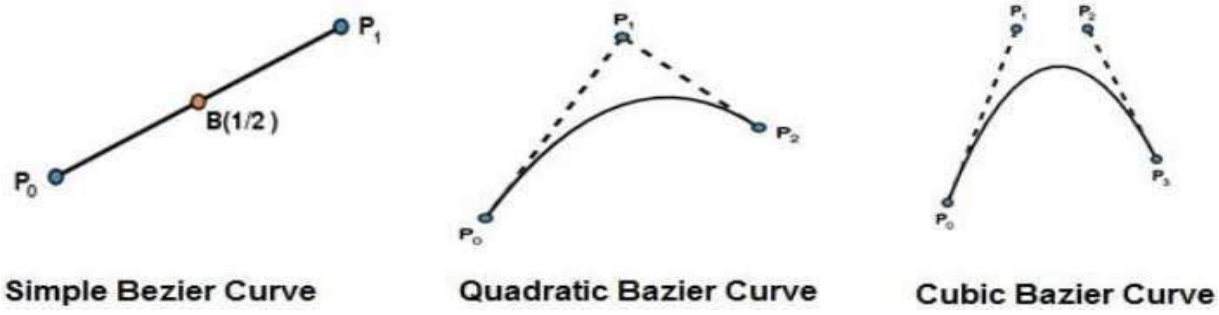

Figure 3.27 Representation of Bezier Curve

Properties of Bezier Curves

Bezier curves have the following properties −

- They generally follow the shape of the control polygon, which consists of the segments joining the control points.
- They always pass through the first and last control points.
- They are contained in the convex hull of their defining control points.
- The degree of the polynomial defining the curve segment is one less that the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial.
- A Bezier curve generally follows the shape of the defining polygon.
- The direction of the tangent vector at the end points is same as that of the vector determined by first and last segments.
- The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
- No straight line intersects a Bezier curve more times than it intersects its control polygon.
- They are invariant under an affine transformation.
- Bezier curves exhibit global control means moving a control point alters the shape of the whole curve.
- A given Bezier curve can be subdivided at a point t=t0 into two Bezier segments which join together at the point corresponding to the parameter value t=t0.

B-Spline Curves Method:

The Bezier-curve produced by the Bernstein basis function has limited flexibility.

- First, the number of specified polygon vertices fixes the order of the resulting polynomialwhich defines the curve.
- The second limiting characteristic is that the value of the blending function is nonzero for allparameter values over the entire curve.

The B-spline basis contains the Bernstein basis as the special case. The B-spline basis is non global.

A B-spline curve is defined as a linear combination of control points Pi and B-spline basis function Ni, k t given by

$$C(t) = \sum_{i=0}^{n} P_i N_{i,k}(t), \qquad n \geq k-1, \qquad t \in [tk-1, tn+1]$$

Where,

- {pi: i=0, 1, 2….n} are the control points

- k is the order of the polynomial segments of the B-spline curve. Order k means that the curve is made up of piecewise polynomial segments of degree k - 1,
- the Ni, k(t) are the "normalized B-spline blending functions". They are described by the order k and by a non-decreasing sequence of real numbers normally called the "knot sequence".

$$ti: i = 0, \ldots n + K$$

The Ni, k functions are described as follows –

$$N_{i,1}(t) = \begin{cases} 1, & if\, u \in [t_i, t_{i+1}) \\ 0, & Otherwise \end{cases}$$

and if k > 1,

$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k-1}} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t)$$

And

$$t \in [t_{k-1}, t_{n+1})$$

Properties of B-spline Curve

- B-spline curves have the following properties –
- The sum of the B-spline basis functions for any parameter value is 1.
- Each basis function is positive or zero for all parameter values.
- Each basis function has precisely one maximum value, except for k=1.
- The maximum order of the curve is equal to the number of vertices of defining polygon.
- The degree of B-spline polynomial is independent on the number of vertices of defining polygon.
- B-spline allows the local control over the curve surface because each vertex affects the shape of a curve only over a range of parameter values where its associated basis function is nonzero.
- The curve exhibits the variation diminishing property.
- The curve generally follows the shape of defining polygon.
- Any affine transformation can be applied to the curve by applying it to the vertices of defining polygon.
- The curve line within the convex hull of its defining polygon.

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in