COMPUTER GRAPHICS – (MODULE – II)

Output Primitives: Line Drawing Algorithms – Loading the Frame Buffer – Line Function – Circle – Generating Algorithms - Attributes of Output Primitives: Line Attributes – Curve Attributes – Color and Gray scale levels– Area fill Attributes – Character Attributes – Bundled Attributes – Inquiry Functions.

Output Primitives

Points and Lines

- Point plotting is done by converting a single coordinate position furnished by an application program into appropriate operations for the output device in use.
- Line drawing is done by calculating intermediate positions along the line path between two specified endpoint positions.
- The output device is then directed to fill in those positions between the end points with some color.
- For some device such as a pen plotter or random scan display, a straight line can be drawn smoothly from one end point to other.
- Digital devices display a straight line segment by plotting discrete points between the two endpoints.
- Discrete coordinate positions along the line path are calculated from the equation of the line.
- For a raster video display, the line intensity is loaded in frame buffer at the corresponding pixel positions.
- Reading from the frame buffer, the video controller then plots the screen pixels.
- 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 2.1.

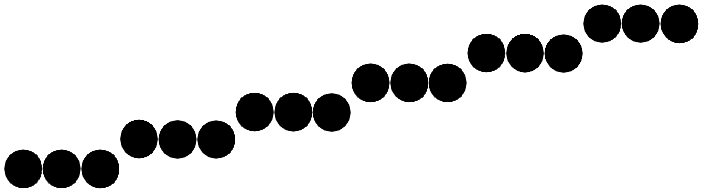


Fig. 2.1: - Stair step effect produced when line is generated as a series of pixel positions.

- The stair step shape is noticeable in low resolution system, and we can improve their appearance somewhat by displaying them on high resolution system.
- More effective techniques for smoothing raster lines are based on adjusting pixel intensities along the line paths.
- For raster graphics device-level algorithms discuss here, object positions are specified directly in integer device coordinates.
- Pixel position will referenced according to scan-line number and column number which is illustrated by following figure.

Fig. 2.2: - Pixel positions referenced by scan-line number and column number.

- To load the specified color into the frame buffer at a particular position, we will assume we have available low-level procedure of the form $setpixel(x, y)$
- Similarly for retrieve the current frame buffer intensity we assume to have procedure $getpix(x, y)$.

Line Drawing Algorithms

The Line drawing algorithm is a graphical algorithm which is used to represent the line segment on discrete graphical media, i.e., printer and pixel-based media.

A line contains two points. The point is an important element of a line.

Properties of a Line Drawing Algorithm

There are the following properties of a good Line Drawing Algorithm.

- **An algorithm should be precise:** Each step of the algorithm must be adequately defined.
- **Finiteness:** An algorithm must contain finiteness. It means the algorithm stops after the execution of all steps.
- **Easy to understand:** An algorithm must help learners to understand the solution in a more natural way.
- **Correctness:** An algorithm must be in the correct manner.
- **Effectiveness:** The steps of an algorithm must be valid and efficient.
- **Uniqueness:** All steps of an algorithm should be clearly and uniquely defined, and the result should be based on the given input.
- **Input:** A good algorithm must accept at least one or more input.
- **Output:** An algorithm must generate at least one output.

Equation of the straight line

We can define a straight line with the help of the following equation.

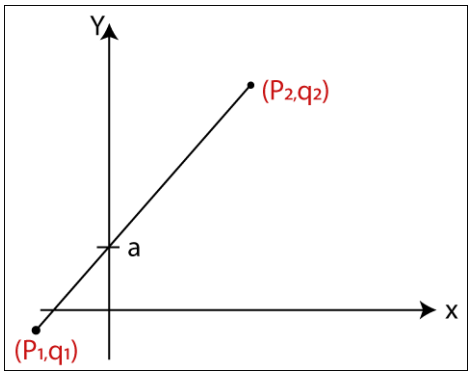
y= mx + a

Where,

(x, y) = axis of the line.

m = Slope of the line.

a = Interception point



Let us assume we have two points of the line $(\mathbf{p_1, q_1})$ and $(\mathbf{p_2, q_2})$.
Now, we will put values of the two points in straight line equation, and we get

$$\mathbf{y = mx + a}$$
$$\mathbf{q_2 = mp_2} \quad \textbf{…(1)}$$
$$\mathbf{q_1 = mp_1 + a} \quad \textbf{…(2)}$$

We have from equation (1) and (2)

$$\mathbf{q_2 - q_1 = mp_2 - mp_1}$$
$$\mathbf{q_2 - q_1 = m\,(p_2 - p_1)}$$

The value of $\mathbf{m = (q_2 - q_1)/ (p_2 - p_1)}$

$$\mathbf{m = \blacktriangle q \,/\, \blacktriangle p}$$

Algorithms of Line Drawing

There are following algorithms used for drawing a line:

- **DDA (Digital Differential Analyzer) Line Drawing Algorithm**
- **Bresenham's Line Drawing Algorithm**
- **Mid-Point Line Drawing Algorith**

DDA (Digital Differential Analyzer) Line Drawing Algorithm

The Digital Differential Analyzer helps us to interpolate the variables on an interval from one point to another point. We can use the digital Differential Analyzer algorithm to perform rasterization on polygons, lines, and triangles.

Digital Differential Analyzer algorithm is also known as an **incremental method** of scan conversion. In this algorithm, we can perform the calculation in a step by step manner. We use the previous step result in the next step.

As we know the general equation of the straight line is:

$$\mathbf{y = mx + c}$$

Here, **m** is the slope of $\mathbf{(x_1, y_1)}$ and $\mathbf{(x_2, y_2)}$.

$$\mathbf{m = (y_2 - y_1)/ (x_2 - x_1)}$$

Now, we consider one point $\mathbf{(x_k, y_k)}$ and $\mathbf{(x_{k+1}, y_{k+1})}$ as the next point.
Then the slope $\mathbf{m = (y_{k+1} - y_k)/ (x_{k+1} - x_k)}$

Now, we have to find the slope between the starting point and ending point. There can be following three cases to discuss:

Case 1: If **m < 1**

Then **x** coordinate tends to the Unit interval.

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

$y_{k+1} = y_k + m$

Case 2: If **m > 1**

Then **y** coordinate tends to the Unit interval.

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

$x_{k+1} = x_k + 1/m$

Case 3: If **m = 1**

Then **x and y** coordinate tend to the Unit interval.

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

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

We can calculate all intermediate points with the help of above three discussed cases.

Algorithm of Digital Differential Analyzer (DDA) Line Drawing

Step 1: Start.

Step 2: We consider Starting point as (x_1, y_1), and endingpoint (x_2, y_2).

Step 3: Now, we have to calculate ▲**x** and ▲**y.**

$\blacktriangle x = x_2 - x_1$

$\blacktriangle y = y_2 - y_1$

$m = \blacktriangle y / \blacktriangle x$

Step 4: Now, we calculate three cases.

If **m < 1**

Then **x** change in Unit Interval

y moves with deviation

$(x_{k+1}, y_{k+1}) = (x_k+1, y_k+1)$

If **m > 1**

Then **x** moves with deviation

y change in Unit Interval

$(x_{k+1}, y_{k+1}) = (x_k+1/m, y_k+1/m)$

If **m = 1**

Then **x** moves in Unit Interval

y moves in Unit Interval

$(x_{k+1}, y_{k+1}) = (x_k+1, y_k+1)$

Step 5: We will repeat step 4 until we find the ending point of the line.

Step 6: Stop.

Example: A line has a starting point (1,7) and ending point (11,17). Apply the Digital Differential Analyzer algorithm to plot a line.

Solution: We have two coordinates,

Starting Point = $(x_1, y_1) = (1,7)$

Ending Point = $(x_2, y_2) = (11,17)$

Step 1: First, we calculate ▲**x**, ▲**y** and **m.**

$\blacktriangle x = x_2 - x_1 = 11-1 = 10$

$\blacktriangle y = y_2 - y_1 = 17-7 = 10$

$m = \blacktriangle y / \blacktriangle x = 10/10 = 1$

Step 2: Now, we calculate the number of steps.

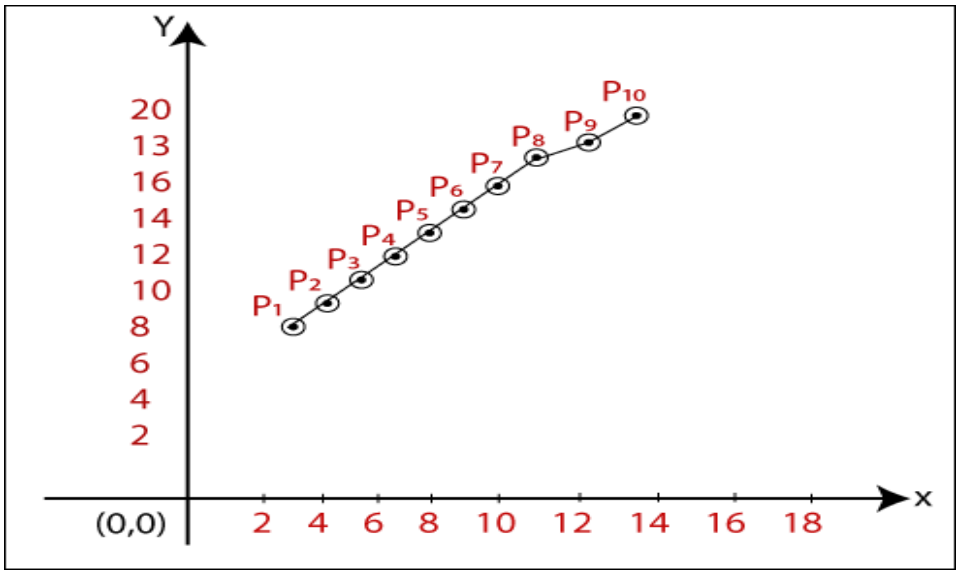
$\blacktriangle x = \blacktriangle y = 10$

Then, the number of steps = **10**

Step 3: We get **m = 1,** Third case is satisfied.
Now move to next step.

Xk	yk	xk+1	yk+1	(xk+1, yk+1)
1	7	2	8	(2, 8)
		3	9	(3, 9)
		4	10	(4, 10)
		5	11	(5, 11)
		6	12	(6, 12)
		7	13	(7, 13)
		8	14	(8, 14)
		9	15	(9, 15)
		10	16	(10, 16)
		11	17	(11, 17)

Step 4: We will repeat step 3 until we get the endpoints of the line.
Step 5: Stop.



The coordinates of drawn line are-
P_1 = (2, 8)
P_2 = (3, 9)
P_3 = (4, 10)
P_4 = (5, 11)
P_5 = (6, 12)
P_6 = (7, 13)
P_7 = (8, 14)
P_8 = (9, 15)
P_9 = (10, 16)
P_{10} = (11, 17)

Advantages of Digital Differential Analyzer

- It is a simple algorithm to implement.
- It is a faster algorithm than the direct line equation.
- We cannot use the multiplication method in Digital Differential Analyzer.
- Digital Differential Analyzer algorithm tells us about the overflow of the point when the point changes its location.

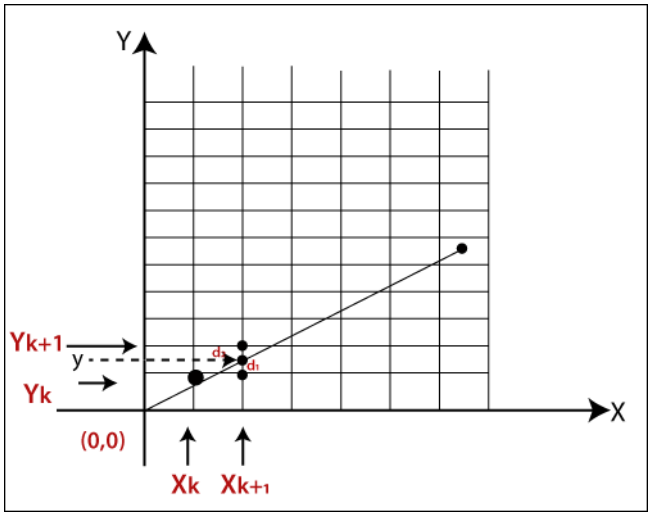
Disadvantages of Digital Differential Analyzer

- The floating-point arithmetic implementation of the Digital Differential Analyzer is time-consuming.
- The method of round-off is also time-consuming.
- Sometimes the point position is not accurate.

Bresenham's Line Drawing Algorithm

This algorithm was introduced by **"Jack Elton Bresenham"** in **1962.** This algorithm helps us to perform scan conversion of a line. It is a powerful, useful, and accurate method. We use incremental integer calculations to draw a line. The integer calculations include addition, subtraction, and multiplication.

In Bresenham's Line Drawing algorithm, we have to calculate the slope (**m**) between the starting point and the ending point.



As shown in the above figure let, we have initial coordinates of a line = **(x_k, y_k).** The next coordinates of a line **= (x_{k+1}, y_{k+1}).** The intersection point between y_k and y_{k+1} = **y.** Let we assume that the distance between **y** and y_k **= d_1.** The distance between **y** and y_{k+1} **= d_2.** Now, we have to decide which point is nearest to the intersection point.

If **m < 1** then **x** = x_k**+1 {** Unit Interval**}**
y = y_k**+1 {** Unit Interval**}**
As we know the equation of a line-
y = mx +b
Now we put the value of x into the line equation, then
$y = m(x_k+1) +b$.....(1)
The value of $d_1 = y - y_k$

Now we put the value of d_1 in equation (1).

Algorithm of Bresenham's Line Drawing Algorithm

Step 1: Start.
Step 2: Now, we consider Starting point as (x_1, y_1) and ending point (x_2, y_2).
Step 3: Now, we have to calculate $\blacktriangle x$ and $\blacktriangle y$.

$\blacktriangle x = x_2 - x_1$

$\blacktriangle y = y_2 - y_1$

$m = \blacktriangle y / \blacktriangle x$

Step 4: Now, we will calculate the decision parameter p_k with following formula.

$p_k = 2\blacktriangle y - \blacktriangle x$

Step 5: The initial coordinates of the line are (x_k, y_k), and the next coordinates are (x_{k+1}, y_{k+1}). Now, we are going to calculate two cases for decision parameter p_k

Case 1: If

$p_k < 0$

Then

$p_{k+1} = p_k + 2\blacktriangle y$

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

$y_{k+1} = y_k$

Case 2: If

$p_k >= 0$

Then

$p_{k+1} = p_k + 2\blacktriangle y - 2\blacktriangle x$

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

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

Step 6: We will repeat step 5 until we found the ending point of the line and the total number of iterations = $\blacktriangle x - 1$.
Step 7: Stop.

Example: A line has a starting point (9,18) and ending point (14,22). Apply the Bresenham's Line Drawing algorithm to plot a line.

Solution: We have two coordinates,
Starting Point = (x_1, y_1) = **(9,18)**
Ending Point = (x_2, y_2) = **(14,22)**

Step 1: First, we calculate $\blacktriangle x$, $\blacktriangle y$.

$\blacktriangle x = x_2 - x_1 = 14 - 9 = 5$

$\blacktriangle y = y_2 - y_1 = 22 - 18 = 4$

Step 2: Now, we are going to calculate the decision parameter (p_k)

$p_k = 2\blacktriangle y - \blacktriangle x$

$= 2 \times 4 - 5 = 3$

The value of $p_k = 3$

Step 3: Now, we will check both the cases.

If $p_k >= 0$ Then **Case 2** is satisfied.

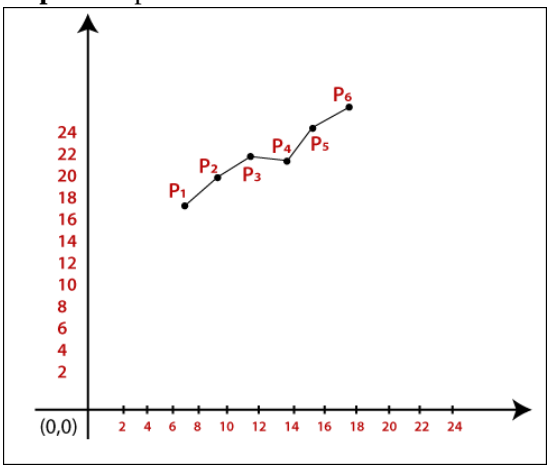
Thus

$$p_{k+1} = p_k + 2\blacktriangle y - 2\blacktriangle x = 3 + (2 \times 4) - (2 \times 5) = 1$$
$$x_{k+1} = x_k + 1 = 9 + 1 = 10$$
$$y_{k+1} = y_k + 1 = 18 + 1 = 19$$

Step 4: Now move to next step. We will calculate the coordinates until we reach the end point of the line.
▲x -1 = 5 – 1 = 4

p_k	p_{k+1}	x_{k+1}	y_{k+1}
		9	18
3	1	10	19
1	-1	11	20
-1	7	12	20
7	5	13	21
5	3	14	22

Step 5: Stop.



The Coordinates of drawn lines are-
P_1 = (9, 18)
P_2 = (10, 19)
P_3 = (11, 20)
P_4 = (12, 20)
P_5 = (13, 21)
P_6 = (14, 22)

Advantages of Bresenham's Line Drawing Algorithm

- It is simple to implement because it only contains integers.
- It is quick and incremental

- It is fast to apply but not faster than the Digital Differential Analyzer (DDA) algorithm.
- The pointing accuracy is higher than the DDA algorithm.

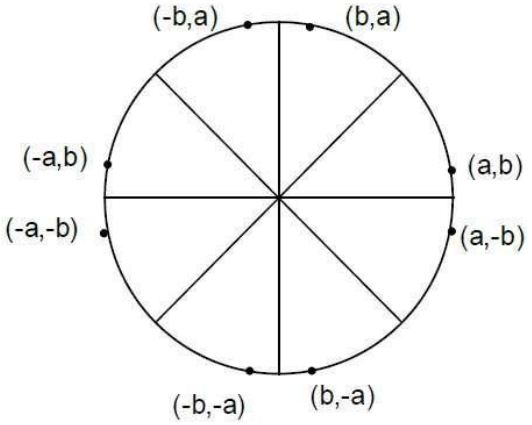
Disadvantages of Bresenham's Line Drawing Algorithm

- The Bresenham's Line drawing algorithm only helps to draw the basic line.
- The resulted draw line is not smooth.

Circle Drawing 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 −

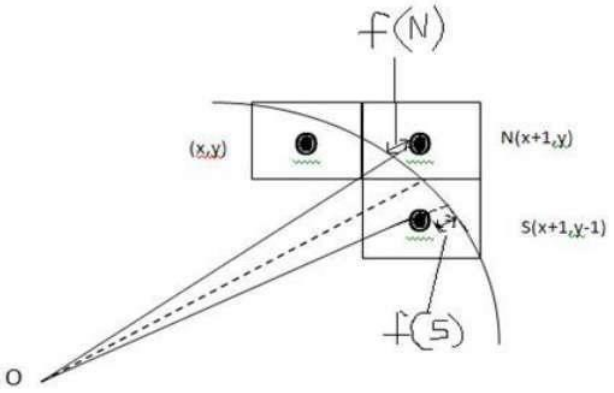
The equation of circle is X2+Y2=r2,X2+Y2=r2, where r is radius.



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,YX,Y location and now need to decide where to put the next pixel − at N X+1,YX+1,Y or at S X+1,Y−1X+1,Y−1.



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

- If d <= 0, then NX+1,YX+1,Y is to be chosen as next pixel.
- If d > 0, then SX+1,Y−1X+1,Y−1 is to be chosen as the next pixel.

Midpoint Algorithm

Step-01:

Assign the starting point coordinates (X0, Y0) as-

X0 = 0

Y0 = R

Step-02:

Calculate the value of initial decision parameter P0 as-

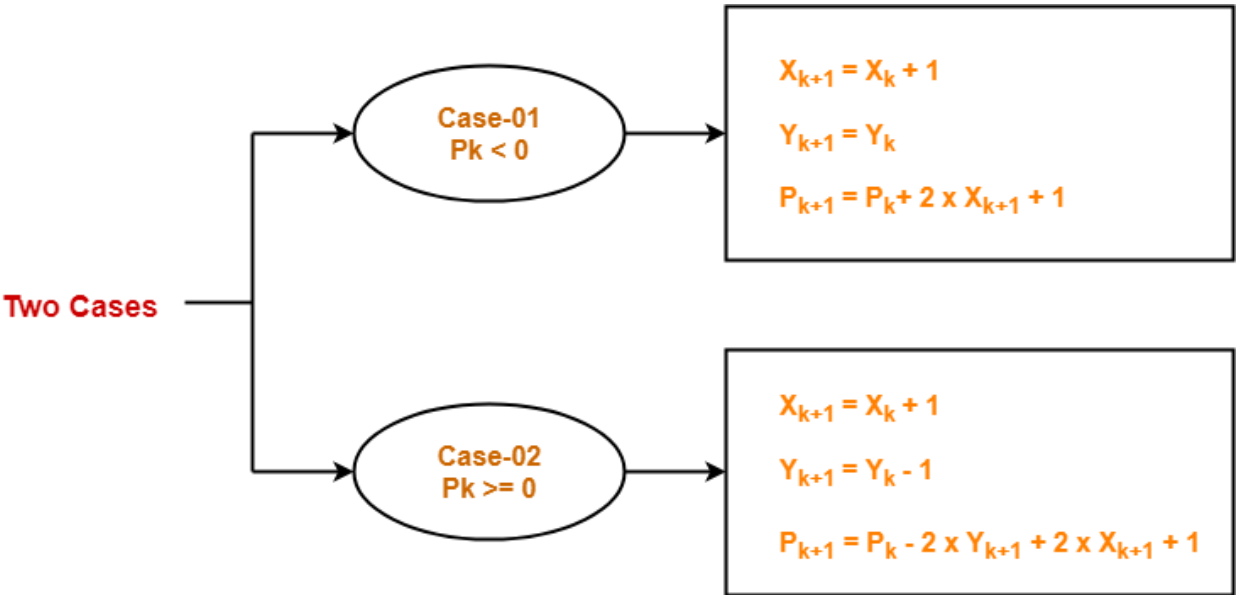
$$P0 = 1 - R$$

Step-03:

Suppose the current point is (Xk, Yk) and the next point is (Xk+1, Yk+1).

Find the next point of the first octant depending on the value of decision parameter Pk.

Follow the below two cases-



Step-04:

If the given centre point (X0, Y0) is not (0, 0), then do the following and plot the point-

Xplot = Xc + X0

Yplot = Yc + Y0

Here, (Xc, Yc) denotes the current value of X and Y coordinates.

Step-05:

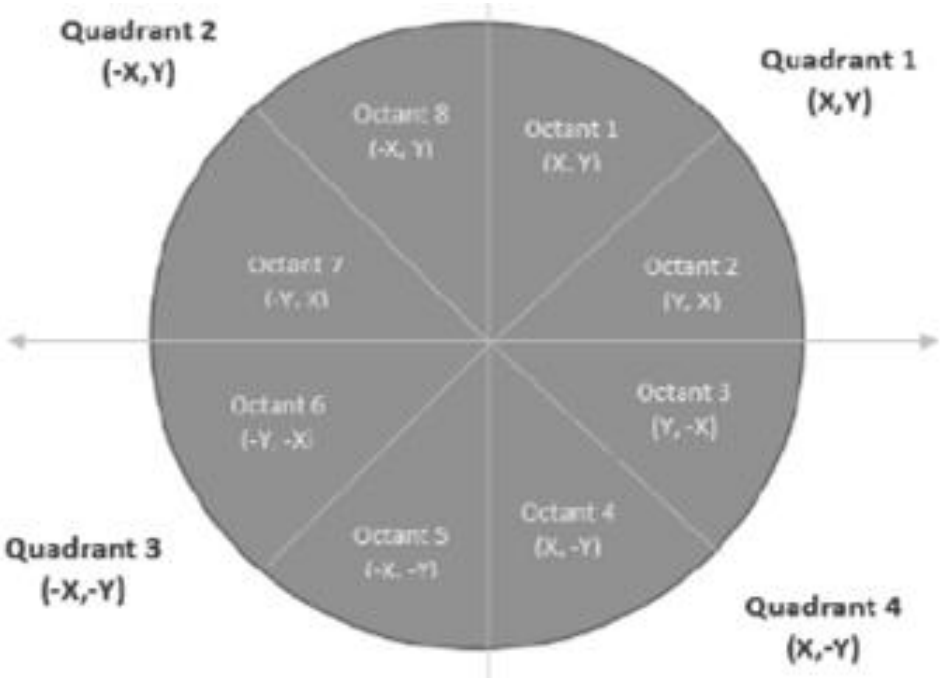
Keep repeating Step-03 and Step-04 until Xplot >= Yplot.

Step-06:

Step-05 generates all the points for one octant.

To find the points for other seven octants, follow the eight symmetry property of circle.

This is depicted by the following figure-



Attributes of output primitives

Attributes

The features or characteristics of an output primitive are known as ***Attribute***. In other words, any parameter that affects the way a primitive is to be displayed is known as ***Attribute***. Some attributes, such as colour and size, are basic characteristics of primitive. Some attributes control the basic display properties of primitives. For example, lines can be dotted or dashed, thin or thick. Areas can be filled with one colour or with multiple colours pattern. Text can appear from left to right, slanted or vertical.

Line Attributes:

Basic attributes of a straight line are its type, its width, and its colour. In some graphics packages, line can also be displayed using selected pen or brush options.

1. **Line Type:** The line type attribute includes solid lines, dashed lines, and dotted lines. We modify the line drawing algorithm to generate such lines by setting the length and space. A dashed line could be displayed by generating spaces that is equal to length of solid part. A dotted line can be displayed by generating very short dashes with spacing equal to or greater than the dash size. Similar methods are used to produce other line-type variations.

Raster line-algorithms displays line type attribute by plotting pixels. For various dashed, dotted patterns, the line-drawing algorithms outputs part of pixels followed by spaces. Plotting dashes with a fixed number of pixels results in unequal-length dashes for different line angles. For example, the length of dash diagonally is more than horizontal dash for same number of pixels. For precision drawings, dash length should remain approximately same for any line angle. For this, we can adjust the pixel number according to line slope.

2. **Line Width:** A line with more width can be displayed as parallel lines on a video monitor. In raster lines, a standard width line is generated with single pixels at each point. Width lines are displayed by plotting additional pixels along next parallel line paths. For lines with slope less than 1, we can display thick lines by plotting a vertical length of pixels at each x position along the line. Similarly, for lines with slope greater than 1, we can plot thick lines with horizontal widths for each point.

The problem with implementing width options using horizontal and vertical pixel widths is that the width of line is depended on the slope. A 45-degree line will be displayed thinner as compared to vertical or horizontal line plotted with same number of pixel widths.

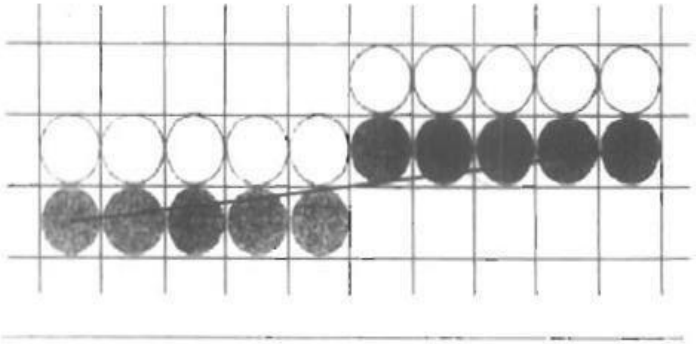


Figure 4-3
Double-wide raster line with slope $|m| < 1$ generated with vertical pixel spans.

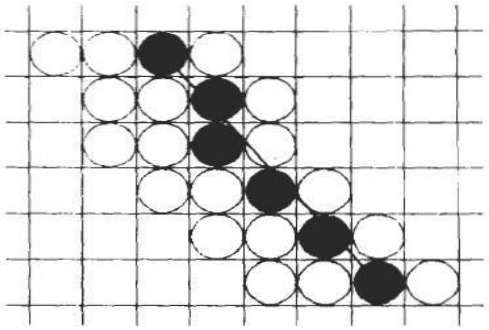
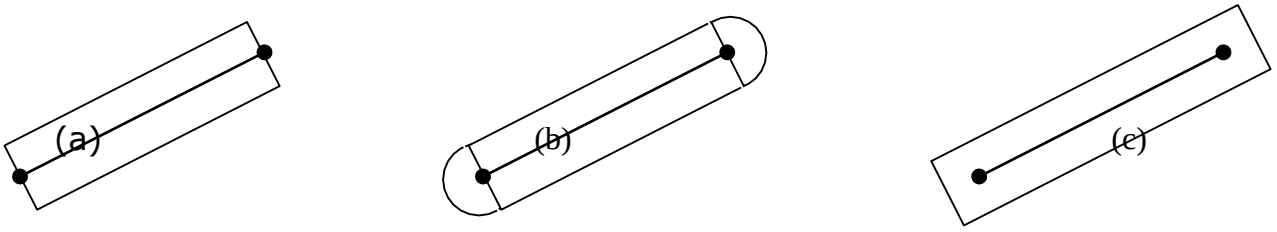


Figure 4-4
Raster line with slope $|m| > 1$ and line-width parameter $lw = 4$ plotted with horizontal pixel spans.

Another problem is that it produces lines whose ends are either horizontal or vertical. We can adjust the shape of the line ends by adding ***Line Caps***.

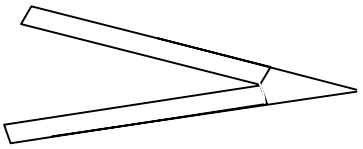
One kind of line cap is the ***Butt Cap.*** It is obtained by adjusting the end positions of lines so that the thick line is displayed with square ends that are perpendicular to the line. Another line cap is the ***Round Cap*** obtained by adding a filled semicircle to each butt cap. The semicircle has diameter equal to thickness of line. The third type of line cap is the ***Projecting Square Cap***. Here, the butt cap is extended to half of line width.



Thick Lines drawn with (a) Butt Cap (b) Round Cap and (c) Projecting Square Cap

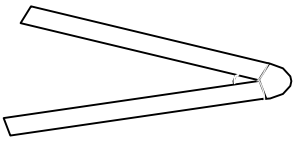
Generating thick connected line segments require other considerations. The methods that we have considered for displaying thick lines will not produce smoothly connected line segments. It leaves gaps at the boundaries between lines of different slope. There are three possible methods for smoothly joining two line segments. A ***Miter Join*** is obtained by extending the outer boundaries of each of the two lines until they meet. A ***Round Join*** is produced by covering the connection between the two segments with a circular boundary whose diameter is equal to the line width. A ***Bevel Join*** is generated by displaying the line with butt caps and filling in the triangular gap where the segments meet.

Line without Joins



Miter Join

Bevel Join



Round Join

3. Pen and Brush Options: In some graphic packages, lines can be displayed with pen or brush selection. Options in this category include shape, size and pattern. These shapes are stored in a ***Pixel Mask*** that identifies the pixel positions that are to be set along the line path. Lines generated with pen or brush shaped can be displayed in various widths by changing the size of the mask. Lines can also be displayed with selected patterns.

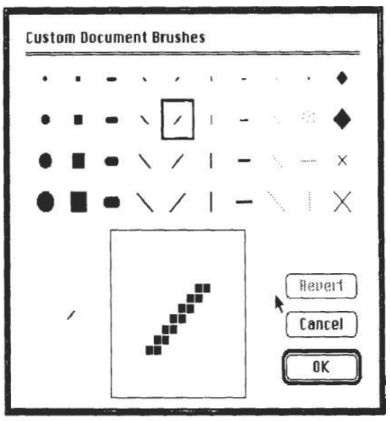


Figure 4-7
Pen and brush shapes for line display.

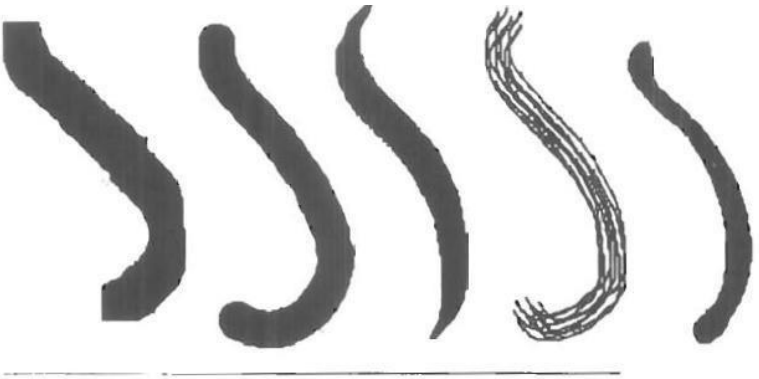


Figure 4-10
Curved lines drawn with a paint program using various shapes and patterns. From left to right, the brush shapes are square, round, diagonal line, dot pattern, and faded airbrush.

4. Line Colour: A system displays a line in the current colour by setting the colour value in the frame buffer at pixel locations. The number of colour choices depends on the number of bits available per pixel in the frame buffer. A line drawn in the background colour is invisible.

COLOR AND GRAYSCALE LEVELS

Various color and intensity-level options can be made available to a user, depending on the capabilities and design objectives of a particular system. General purpose raster-scan systems, for example, usually provide a wide range of colors, while random-scan monitors typically offer only a few color choices, if any. Color options are numerically coded with values ranging from 0 through the positive integers. For CRT monitors, these color codes are then converted to intensity level settings for the electron beams. With color plotters, the codes could control ink-jet deposits or pen selections.

In a color raster system, the number of color choices available depends on the amount of storage provided per pixel in the frame buffer. Also, color-information can be stored in the frame buffer in two ways: We can store color codes directly in the frame buffer, or we can put the color codes in a separate table and use pixel values as an index into this table. With the direct storage scheme, whenever a particular color code

is specified in an application program, the corresponding binary value is placed in the frame buffer for each-component pixel in the output primitives to be displayed in that color. A minimum number of colors can be provided in the scheme with 3 bits of storage per pixel, as shown in Table. Each of the three bit positions is used to control the intensity level (either on or off) of the corresponding electron gun in an RGB monitor. The leftmost bit controls the red gun, the middle bit controls the green gun, and the rightmost bit controls the blue gun. Adding more bits per pixel to the frame buffer increases the number of color choices. With 6 bits per pixel, 2 bits can be used for each gun. This allows four different intensity settings for each of the three color guns, and a total of 64 color values are available for each screen pixel. With a resolution of 1024 by 1024, a full-color (24bit per pixel) RGB system needs 3 megabytes of storage for the frame buffer. Color tables are an alternate means for providing extended color capabilities to a user without requiring large frame buffers. Lower-cost personal computer systems, in particular, often use color tables to reduce frame-buffer storage requirements.

COLOR TABLES

Figure 2.1 illustrates a possible scheme for storing color values in a color lookup table (or video lookup table), where frame-buffer values are now used as indices into the color table. In this example, each pixel can reference any one of the 256 table positions, and each entry in the table uses 24 bits to specify an RGB color. For the color code 2081, a combination green-blue color is displayed for pixel location (x, y). Systems employing this particular lookup table would allow a user to select any 256 colors for simultaneous display from a palette of nearly 17 million colors. Compared to a full color system, this scheme reduces the number of simultaneous colors that can be displayed,

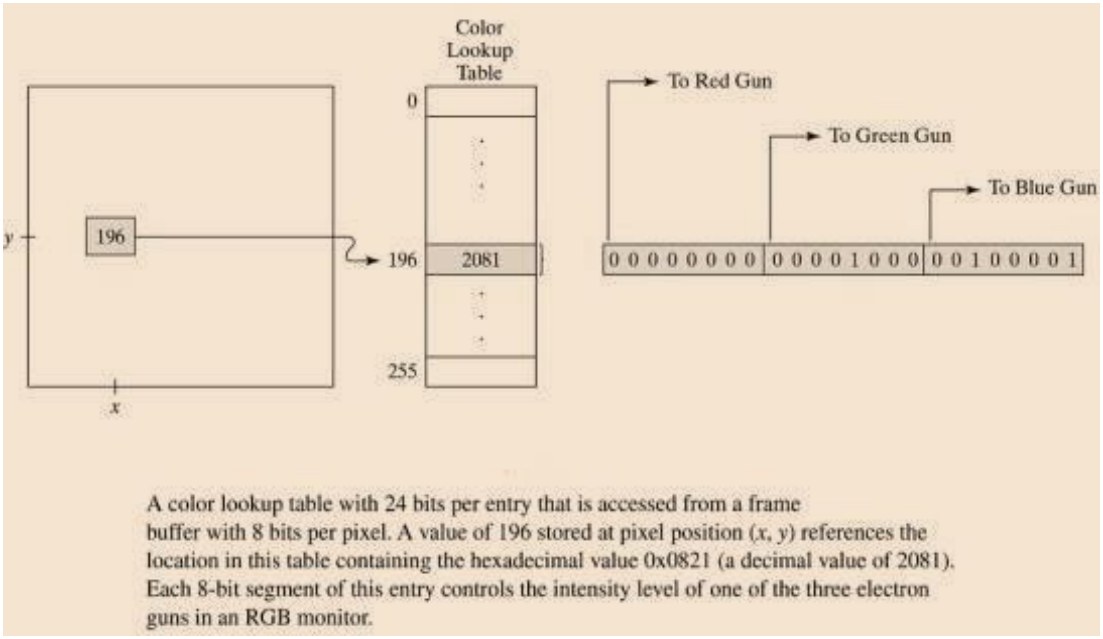
but it also reduces the frame buffer storage requirements to 1 megabyte. Some graphics systems provide 9 bits per pixel in the frame buffer, permitting a user to select 512 colors that could be used in each display.

TABLE 4-1

THE EIGHT RGB COLOR CODES FOR A THREE-BIT PER PIXEL FRAME BUFFER

	Stored Color Values in Frame Buffer			
Color Code	RED	GREEN	BLUE	Displayed Color
0	0	0	0	Black
1	0	0	1	Blue
2	0	1	0	Green
3	0	1	1	Cyan
4	1	0	0	Red
5	1	0	1	Magenta
6	1	1	0	Yellow
7	1	1	1	White

(Table 2.1, The eight color codes for a three-bit per pixel frame buffer)



A color lookup table with 24 bits per entry that is accessed from a frame buffer with 8 bits per pixel. A value of 196 stored at pixel position (x, y) references the location in this table containing the hexadecimal value 0x0821 (a decimal value of 2081). Each 8-bit segment of this entry controls the intensity level of one of the three electron guns in an RGB monitor.

(Fig: 2:1, Color Lookup Table)

There are several advantages in storing color codes in a lookup table. Use of a color table can provide a "reasonable" number of simultaneous colors without requiring large frame buffers. For most applications, 256 or 512 different colors are sufficient for a single picture.

GRAYSCALE

With monitors that have no color capability, color functions can be used in an application program to set the shades of gray, or grayscale, for displayed primitives. Numeric values over the range from 0 to 1 can be used to specify grayscale levels, which are then converted to appropriate binary codes for storage in the raster. This allows the intensity settings to be easily adapted to systems with differing grayscale capabilities.

Table lists the specifications for intensity codes for a four-level grayscale system. In this example, any intensity input value near 0.33 would be stored as the binary value 01 in the frame buffer, and pixels with this value would be displayed as dark gray. If additional bits per pixel are available in the frame buffer, the value of 0.33 would be mapped to the nearest level. With 3 bits per pixel, we can accommodate 8 gray

levels; while 8 bits per pixel would give us 256 shades of gray. An alternative scheme for storing the intensity information is to convert each intensity code directly to the voltage value that produces this grayscale level on the output device in use.

INTENSITY CODES FOR A FOUR-LEVEL GRAYSCALE SYSTEM

Intensity	Stored Intensity Codes Values In the frame Buffer (Binary Code)		Displayed Gray Scale
0.0	0	(00)	Black
0.33	1	(01)	Dark gray
0.67	2	(1 0)	Light gray
1 .O	3	(11)	White

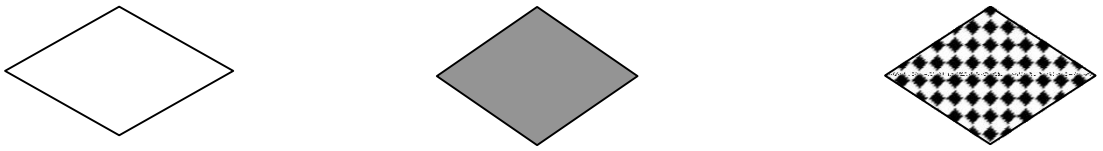
Area-Fill Attributes:

Options for filling a region include a choice between a solid colour and a patterned fill. These options can be applied to polygon regions or regions with curved boundaries.

Areas can be displayed with various fill styles: hollow, solid, pattern and hatch.

Hollow areas are displayed using only the boundary outline, with interior colour the same as the background colour.

A Solid fill is displayed in a single colour including the borders. Fill style can also be with a specific pattern or design. The Hatch fill is used to fill area with hatching pattern.



Hollow Solid Pattern

Polygon fill styles

Diagonal hatch Diagonal Cross-hatch

Polygon fill using hatch patterns

Other fill options include specifications for the edge type, edge width and edge colour. These attributes are same as the line attributes. That is, we can display polygon edges as dotted, dashed, thick or of different colours, etc.

Soft Fill

We may want to fill area again due to 2 reasons:

- It is blurred (unclear) when painted first time, or
- It is repainting of a color area that was originally filled with semitransparent brush, where current color is then mixture of the brush color and the background color "behind" the area.

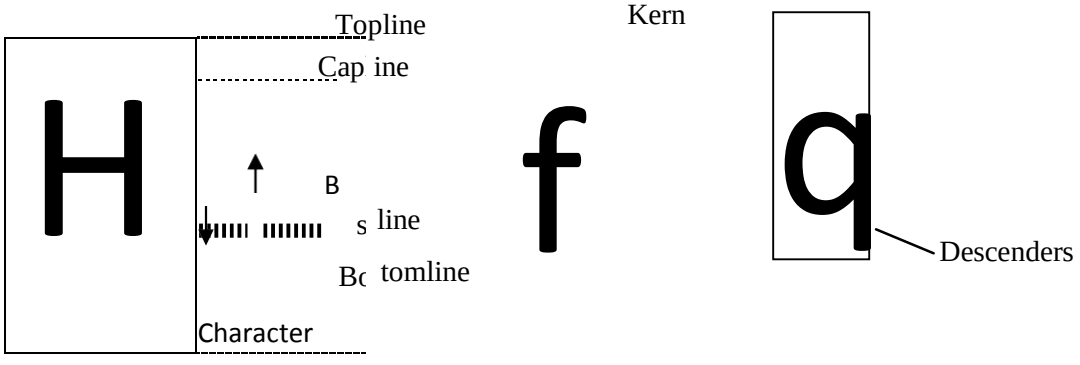
So that the fill color is combined with the background colors are referred to as Soft-fill.

Character Attributes:

The appearance of displayed characters is controlled by attributes such as font, size, colour and orientation. Attributes can be set both for entire character strings and for individual characters, known as Marker symbols.

1. Text Attributes: There are many text options available, such as font, colour, size, spacing, and orientation.

- **Text Style:** The characters in a selected font can also be displayed in various underlining styles (solid, dotted, dashed, double), in **bold**, in *italics*, shadow style, etc. Font options can be made available as predefined sets of grid patterns or as character sets designed with lines and curves.
- **Text Colour:** Colour settings for displayed text are stored in the system attribute list and transferred to the frame buffer by character loading functions. When a character string is displayed, the current colour is used to set pixel values in the frame buffer corresponding to the character shapes and position.
- **Text Size:** We can adjust text size by changing the overall dimensions, i.e., width and height, of characters or by changing only the width. Character size is specified in ***Points***, where 1 point is 0.013837 inch or approximately 1/72 inch. Point measurements specify the size of the ***Character Body***. Different fonts with the same point specifications can have different character sizes depending upon the design of the font.



Character Body

The distance between ***Topline*** and ***Bottomline*** is same for all characters in a particular size and font, but the width may be different. A smaller body width is assigned to narrow characters such as i, j, l, e t c . compared to broad characters such as W or M. The ***Character Height*** is the distance between the ***Baseline*** and ***Capline*** of characters. Kerned characters, such as f and j, extend beyond the character-width limits. And letters with descenders, such as g, j, p, q, extend below the baseline.

The size can be changed in such a way so that the width and spacing of characters is adjusted to maintain the same text proportions. For example, doubling the height also doubles the character width and the spacing between characters. Also, only the width of the character s can be changes without affecting its height. Similarly, spacing between characters can be increased without changing height or width of individual characters.

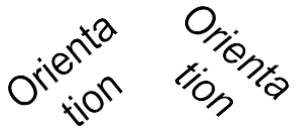
The effect of different character-height setting, character-width setting, and character spacing on a text is shown below.

HEIGHT1	WIDTH1	SPACING1
HEIGHT2	**WIDTH2**	**SPACING2**
HEIGHT3	***WIDTH3***	***S P A C I N G 3***

Effect of changing Height, Width and Spacing

- **Text Orientation:** The text can be displayed at various angles, known as orientation. A procedure for orienting text rotates characters so that the sides of character bodies, from baseline to topline at aligned at some angle. Character strings can be arranged vertically or horizontally.

Orientation



A text orientated by 45 degrees in anticlockwise and clockwise direction

- **Text Path:** In some applications the character strings are arranged vertically or horizontally. This is known as Text Path. Text path can be right, left, up or down.

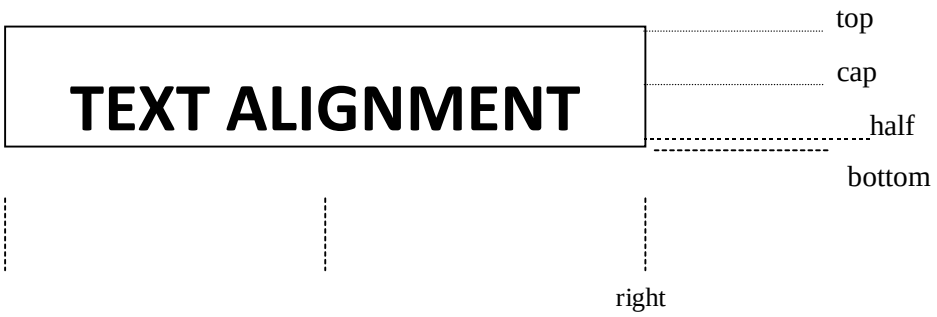
g
n
i
r
t
s

g n i r t s String

s
t
r
i
n
g

A text displayed with the four text-path options

- **Text Alignment:** Another attribute for character strings is alignment. This attribute specifies how text is to be positioned with respect to the start coordinates. Vertical alignment can be top, cap, half, base and bottom. Similarly, horizontal alignment can be left, centre and right.



Alignment values for a string

Text Attributes

- In text we are having so many style and design like italic fonts, bold fonts etc.
- For setting the font style in PHIGS package we have one function which is:

setTextFont (tf)

- Where tf is used to specify text font
- It will set specified font as a current character.
- For setting color of character in PHIGS we have function:

setTextColorIndex (tc)

- Where text color parameter tc specifies an allowable color code.
- For setting the size of the text we use function.

setCharacterheight (ch)

- For scaling the character we use function.

setCharacterExpansionFacter (cw)

- Where character width parameter cw is set to a positive real number that scale the character body width.
- Spacing between character is controlled by function

setCharacterSpacing (cs)

- Where character spacing parameter cs can be assigned any real value.
- The orientation for a displayed character string is set according to the direction of the character up vector:

setCharacterUpVector (upvect)

- Parameter upvect in this function is assigned two values that specify the x and y vector components Text is then displayed so that the orientation of characters from baseline to cap line is in the direction of the up vector.
- For setting the path of the character we use function:

setTextPath (tp)

- Where the text path parameter tp can be assigned the value: right, left, up, or down.
- It will set the direction in which we are writing.
- For setting the alignment of the text we use function.

setTextAlignment (h, v)

- Where parameter h and v control horizontal and vertical alignment respectively.
- For specifying precision for text display is given with function.

setTextPrecision (tpr)

- Where text precision parameter tpr is assigned one of the values: string, char, or stroke.
- The highest-quality text is produced when the parameter is set to the value stroke.

Marker Attributes

- A marker symbol display single character in different color and in different sizes.
- For marker attributes implementation by procedure that load the chosen character into the raster at defined position with the specified color and size.
- We select marker type using function.

setMarkerType (mt)

- Where marker type parameter mt is set to an integer code.
- Typical codes for marker type are the integers 1 through 5, specifying, respectively, a dot (.), a vertical cross (+), an asterisk (*), a circle (o), and a diagonal cross (x). Displayed marker types are centered on the marker coordinates.
- We set the marker size with function.

SetMarkerSizeScaleFactor (ms)

- Where parameter marker size ms assigned a positive number according to need for scaling.
- For setting marker color we use function.

setPolymarkerColorIndex (mc)

- Where parameter mc specify the color of the marker symbol

Bundled Attributes

A particular set of attribute values tor a primitive on each output device is then chosen by specifying the appropriate table index. Attributes specified in this manner called bundled attributes

Bundled line Attributes

Entries in the bundle table for line attributes on a specified workstation are set with the function

setPolylineRepresentation (ws, li, lt, lw, lc)

Parameter ws is the workstation identifier and line index parameter li defines the bundle table position. Parameter lt, lw, tc are then bundled and assigned values to set the line type, line width, and line color specifications for designated table index.

Example

setPolylineRepresentation (1, 3, 2, 0.5, 1)

setPolylineRepresentation (4, 3, 1, 1, 7)

A poly line that is assigned a table index value of 3 would be displayed using dashed lines at half thickness in a blue color on work station 1; while on workstation 4, this same index generates solid, standard-sized white lines

Bundle area fill Attributes

Table entries for bundled area-fill attributes are set with

setInteriorRepresentation (ws, fi, fs, pi, fc)

Which defines the attributes list corresponding to fill index fi on workstation ws. Parameter fs, pi and fc are assigned values for the fill style pattern index and fill color.

Bundled Text Attributes

setTextRepresentation (ws, ti, tf, tp, te, ts, tc)

Bundles values for text font, precision expansion factor size an color in a table position for work station ws that is specified by value assigned to text index parameter ti.

Bundled marker Attributes

setPolymarkerRepresentation (ws, mi, mt, ms, mc)

That defines marker type marker scale factor marker color for index mi on workstation ws.

Inquiry Function

Current settings for attributes and other parameters as workstations types and status in the system lists can be retrieved with inquiry functions.

inquirePolylineIndex (lastli)

and

inquireInteriorcColourIndex (lastfc)

Copy the current values for line index and fill color into parameter lastli and lastfc.