C.P.PATEL & F.H.SHAH COMMERCE COLLEGE

(MANAGED BY SARDAR PATEL EDUCATION TRUST)

BCA, BBA (ITM) & PGDCA PROGRAMME

BCA SEM V (COMPUTER GRAPHICS)

UNIT 2 – Output Primitives and their attributes

UNIT 2 – Output Primitives and their attributes

- Output Primitives: Points, Lines, Circles
- Line Drawing Algorithm (Without program): Digital Differential Analyzer (DDA) and Bresenham's algorithm
- Circle generating algorithm (without program): Midpoint Circle Algorithm
- Filled Area primitives: Scan Line Polygon Fill Algorithm (Without Program)
- Inside-Outside tests: Odd even rule & Non-zero winding number rule
- Boundary Fill Algorithm (with procedure)
- Flood Fill Algorithm (with procedure)
- Character Generation
- Attributes: Line, Color, Area fill, Character

Output Primitives: Points, Lines, Circles

Point

In random scan, co-ordinate value is converted into the plotting of point is done by converting a single co-ordinate position of a program into the output device, which is in use. In CRT, the electron beam is turned 'on' to glow the phosphor at selected position.

In raster scan with black & white, the point is plotted by making the bit value of that position in frame buffer to 1.

Line

A line can be drawn by calculating the intermediate points of the two end points. In random scan, the straight line can be drawn very smoothly, by changing the horizontal and vertical deflection voltages. In raster scan, the color of line (intensity) for a pixel position is stored in the frame buffer.

The intermediate points of line are in terms of the integer. So the value obtained is rounded, and point is plotted.

Circle

A circle is defined as the "Set points that are all at a given distance r from the centre position (Xc, Yc)".

The circle is used many times in the pictures and graphs, and so the procedures for drawing circle or arcs are provided in most of the graphics packages. Generally, the procedures can draw either circular or elliptical curves.

The distance relationship is expressed by the Pythagorean Theorem in Cartesian coordinates as:

$$(x-xc)^2+(y-yc)^2=r^2$$

In terms of the polar co-ordinates the equations of x and y points of circle are given as:

$x = xc + r \cos\theta$

$Y = Yc + r \sin\theta$

The points around the centre can be calculated by the algorithms like Bresenham's, and the circle can be drawn accurately. The midpoint algorithm creates circle using 8 octants as below. The points obtained in this algorithm are same as that in Bresenham's.

Line Drawing Algorithm:

The Cartesian slope intercept equation for a straight line is

$$y = m.\ x + b \qquad (3\text{-}1)$$

With m representing the slope of the line and b as the y intercept. Given that the two endpoints of a line segment are specified at positions (x_1, y_1) and (x_2, y_2), as shown in Fig: 3-3, we can determine values for the slope m and y intercept b with the following equations:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \qquad (3\text{-}2)$$

$$b = y_1 - m \cdot x_1 \qquad (3\text{-}3)$$

Algorithms for displaying straight lines are based on the line equation 3-1 and the calculations given in Eqs: 3-2 and 3-3.

For any given x interval Δx along a line, we can compute the corresponding y interval Δy from Eq. 3-2 as

Δy = mΔx ---------- (3.4)

Similarly, we can obtain the x interval Δx corresponding to a specified Δy as

Δx = Δy / m ---------- (3.5)

These equations form the basis for determining deflection voltages in analog devices.

There are basically two major algorithms used for line drawing, using above eqns., as follows.

Digital Differential Analyzer (DDA) Algorithm:

The Digital Difference Analyser (DDA) algorithm is faster method for calculating pixel positions. It is based on the either calculating either Δx **or Δy** The line to be drawn may have slope as positive or negative. Positive, slope means that values of x and y is increasing as in below line:

If the positive value of slope is less or equal to 1, the value of next point y is given as:

yk+1=yk+m --(I)

Here. k starts from 1 and increases by 1 until endpoint is obtained. The value of y points is real no., and sr' we have to convert it into integer.

If the positive value of slope is greater than 1, we calculate the values of x given by eqn.

Xk + 1/ m —- (ii)

The above two eqns. are used only for line from left to right endpoints. For the line from right to left, the values are decremented, and thus eqns, are:

Yk+1 = Yk - m -- (iii)

xk+1=xk-1/m --(iv)

Similarly, for the negative slope, and less or equal to 1, eqn. (i) and (iii) are used from left to right, and for greater than 1, eqn. (ii) and (iv) are used from right to left.

From the above rules, we can summarise the algorithm as below:

1. dx = xb - xa; dy = yb - ya [the difference of endpoints]

2. if abs(dx) > abs(dy) then steps = abs(dx) else steps = abs(dy)

3. Xincrement = dx / steps ; yincrement = dy / steps [Find total intermediate' points]

4. setPixel (round(xa),round(ya), 1) [the first point]

5. Repeat step 6 steps times

6. x = x + xincrement ; y = y + yincrernent; setPixel(round(x), round(y), 1) [all pt.s]

Further more, we can improve the performance of the DDA algorithm by separating the increments **m** and **1/m** into integer and fractional parts so that all calculations are reduced to integer operations.

Bresenham's Line Drawing algorithm:

It was developed by Bresenham, and it is accurate and efficient algorithm to display not only lines, but also the circles and other curves. The integer values of the co-ordinates of next point in line are obtained by the decision parameters, as per following steps, and thus the line is created.

1. Input the two-line endpoints and store the left endpoint (x,y)

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

3. Calculate the constants Δx, Δy, 2Δy and 2Δy - 2Δx, and obtain the first decision parameter as Po = 2Δy – Δx

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

If Pk < 0, the next point to plot is (Xk + 1, yk) and next parameter

$$Pk+1 = Pk + 2\Delta y$$

Otherwise, the next point to plot is (Xk + 1, Yk + 1), and next parameter =

$$Pk+1 = Pk + 2\Delta y - 2\Delta x$$

5. Repeat step 4, ΔX no. of times.

Circle generating algorithm: Midpoint Circle Algorithm

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

$$(x_0, Y_0) = (0, r)$$

2. Calculate the initial value of decision parameter as

$$P_0 = 5/4 - r$$

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

If Pk < 0, the next point along circle centered on (0,0) is (Xk+l, Yk) and

$$P_{k+1} = Pk + 2x_{k+1} + 1$$

Otherwise the next point is (x + I Y - i) and

$$P_{k+1} = Pk + 2x_{k+1} + 1 - 2y_{k+1}$$

where $2x_{k+1} = 2x_k + 2$ and $2y_{k+1} = 2y_k - 2$.

4. Determine the 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 coordinated values:

$$x = x + x_c$$

$$y = y + y_c$$

6. Repeat steps 3 through 5 until x>=y.

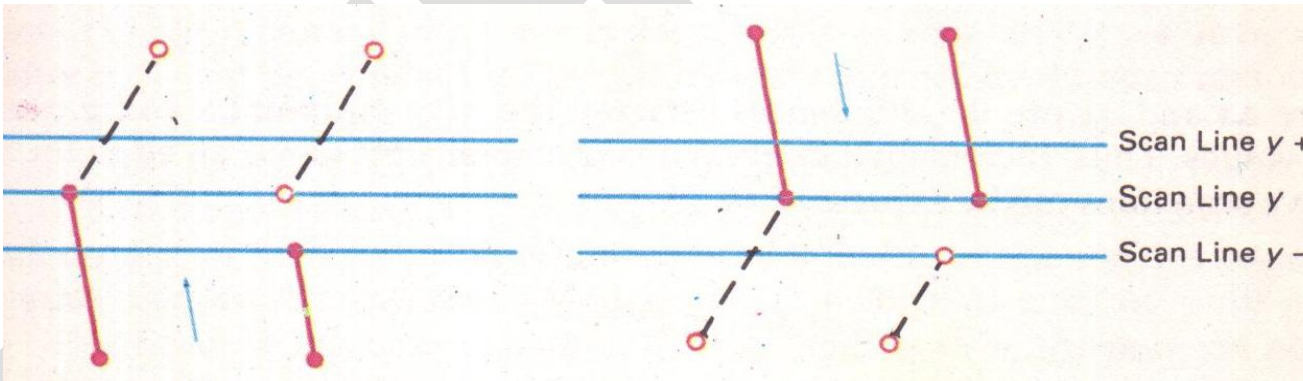
Filled Area primitives: Scan Line Polygon Fill Algorithm

The standard output primitive in graphics packages is a solid-color or polygon area with some pattern. There are two basic approaches for filling the area on raster system. One is to determine the area which is overlapping the scan lines. Another is to start from the internal point and go on painting towards outside until the boundary condition is reached.

In this, for each scan line crossing a polygon, the area-fill algorithm locates the intersection points of the scan line with the polygon edges. These intersection points are then sorted from left to right, and the corresponding frame – buffer positions between each intersection pair are set to the specified fill color.

The filling creates problem when the intersection point contains the vertices (instead of edges). Actually, the vertices are intersection of the two lines, and hence we will consider the intersection of a single vertex point as two edges, as in scan-line ‘b Thus, the total no. of points becomes 1+2+1 = 4. ln this way we can easily fill the areas, whenever we get even no. of intersection points.

The condition like scan-line ‘c’ may also appear where we have 1+2+1+1 = 5 points. Since it is odd no., it is not possible to decide filling areas. In this type of situation, solution is to split the vertex into two parts. The splitting is done by checking whether we (see Fig) scan edges from top or bottom. If it is from bottom. we cut first edge to previous point (y-1), and in scanning from top, we cut second. edge to next point, as in fig(b).



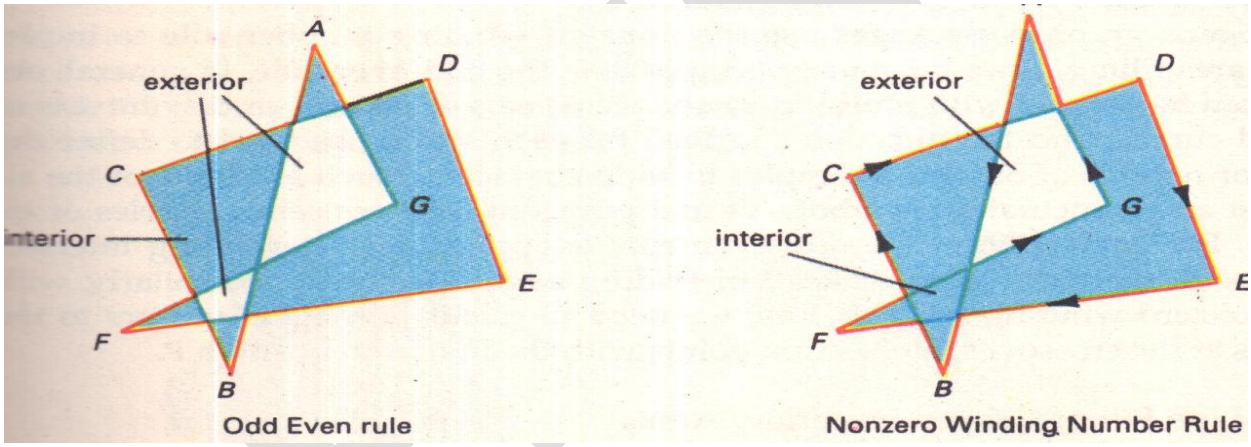
Inside outside Tests

When the area is to be filled, we have to identify the interior points of the object. Now the polygon shapes are generally consisting of more then one vertices, and we assume that it does not intersect with itself. But in some objects as in below fig (3-40 on page no 125)., we may require any area to be interior, though it may be the self-intersection. In most of the graphics packages, the odd-even parity rule or non-zero winding no. is used to find the interior point.

In **odd-even** parity rule, which is also known as **odd parity rule** or **even parity rule**, we draw a line from any position P to a distant point, which is outside the co-ordinate extents of the object and counting the number of edge crossings along the line.. If the total no. of edges that cross (intersect) this line is odd, then point P is interior point. If it is even, then P is exterior. Here, we should take care that the lines do not pass through any polygon vertices. The scan-line polygon fill is example of odd-even parity rule:

In **non-zero** winding no., we find the no. of polygon edges which are winding (surrounding) the given point in counter-clockwise direction. The count obtained is known as **winding number**. The point whose winding no. is non-zero is considered as interior point, and otherwise, it is exterior.

Identifying the interior and exterior points according to both above methods is as shown in below fig.(3.-40 a) and fig (3-40 b).



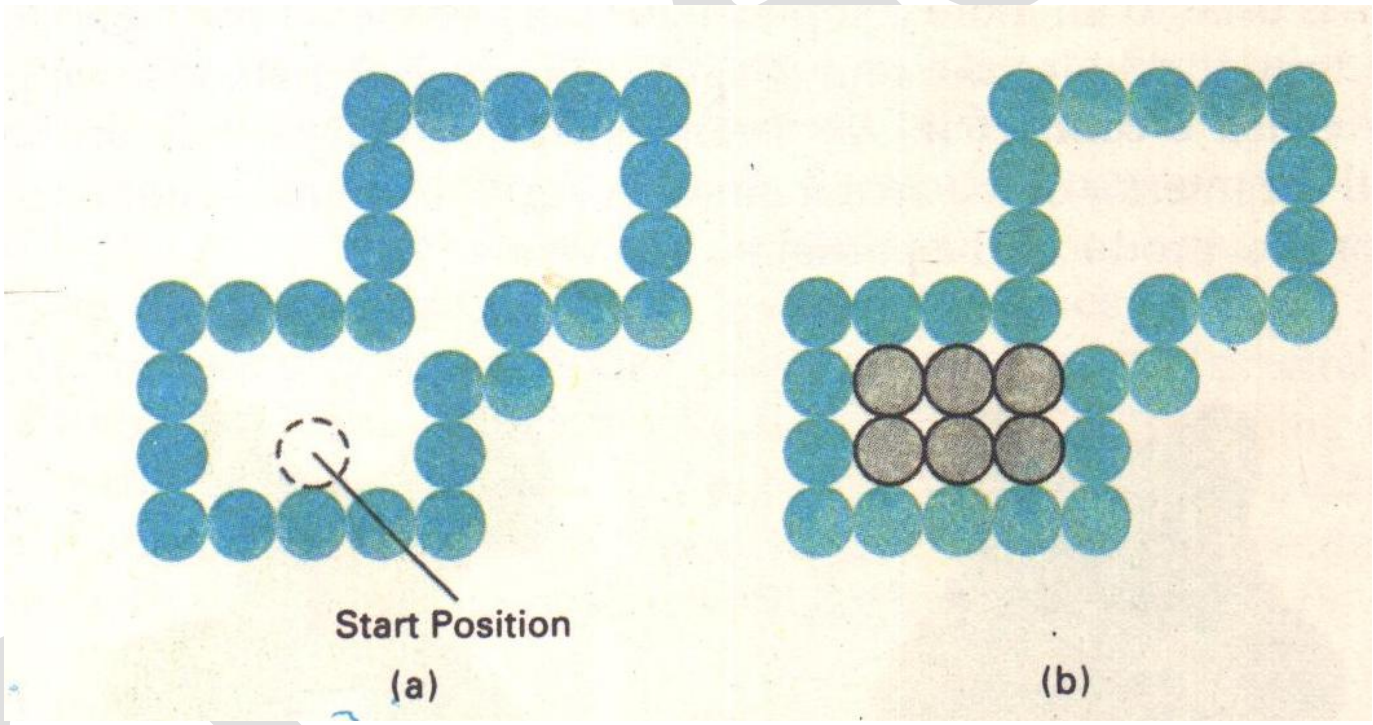
Boundary Fill Algorithm

It is another method of filling the area of a polygon. The painting is started with the internal point, and we move towards the external point until the boundary color is obtained. Thus, the algorithm is known as **boundary fill algorithm**.

If solid-color .is required (no border), then user should choose the fill color same as the boundary color. The boundary fill algorithm takes as input the (x,y) position of the internal point, the fill color and the boundary color. It starts by painting the neighboring pixels with fill color if they do not have boundary color. If pixels are found with boundary color, it will stop.

There are 2 methods for filling area: **4-connected** which uses only left,right, top and bottom pixels, and **8-connected** which uses 8 pixels surrounding, as in below fig.(3-44) The area filled by 8-connected is correct fill, while previous one uses partial fill.

```
procedure boundaryFill(x, y, fill, boundary: integer);
var
  current: integer;
begin
  current = getPixel (x, y);
  if (current <> boundary) and (current <> fill) then
    begin
      setPixel (x, y, fill);
      boundaryFill4 (x+1, y, fill, boundary);
      boundaryFill4 (x-1, y, fill, boundary);
      boundaryFill4 (x, y+1, fill, boundary);
      boundaryFill4 (x, y-1, fill, boundary)
    end
end; { boundaryFill }
```



Flood-Fill Algorithm

In some cases, we may require to fill the area, which does not have a single color boundary, but has several colors. We can replace the old color of interior points by the required fill color using the flood-fill algorithm, as follows. Here we start with one interior point, as we go on painting them with fill color until all the points are considered.

Two approaches are used for flood-filling .**1)4 Connected** **2)8 Connected**

The following procedure flood fills a 4-connected region recursively, starting from the input position.

```
Procedure floodfill4 (x, y, fillcolor, oldcolor: integer);
Begin
        If getpixel (x, y) = oldcolor then
        Begin
                Setpixel(x,y,fillcolor);
                Floodfill4(x+1,y,fillcolor,oldcolor);
                Floodfill4(x-1,y,fillcolor,oldcolor);
                Floodfill4(x,y+1,fillcolor,oldcolor);
                Floodfill4(x,y-1,fillcolor,oldcolor);
        End
End; {floodfill}
```

Character Generation

There are variety of sizes and styles of characters, letters and numbers that we can display. The overall design of set of characters is called **typeface**.

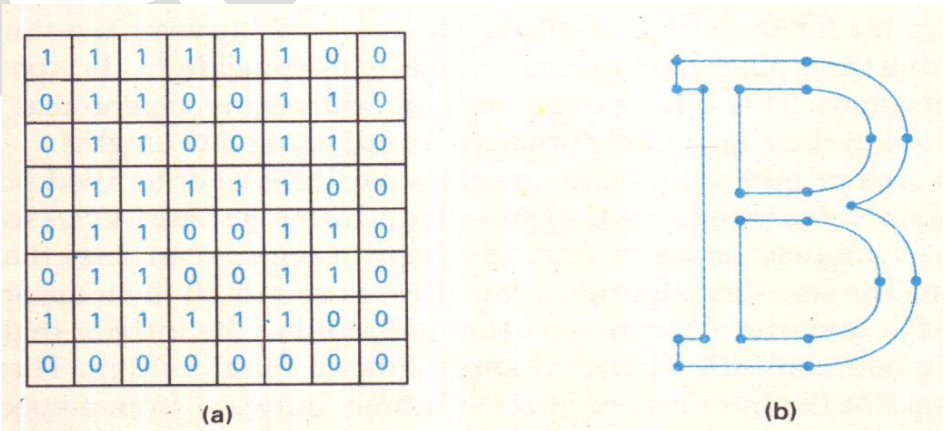
There are hundreds of typefaces available today for computer applications. They are also referred as fonts. It can be classified mainly into two types: **Serif** and **Sans serif**. Serif font consists of the accents (small lines) for each character, while sans serif does not. Generally, serif font is more readable when we have long text, but individual characters of Sans-serif are easier to recognize i.e. legible.

The character can be stored using two different representations. **Bitmap font** uses the frame buffer to store the value for point to be plotted as 1, and others as 0. In the **Outline font**, the character is created by drawing a set of lines. Both methods are shown as below in Fig ((a) and (b))

There are two basic **PHIGS** functions used for the creation of the text:

text (wcPoint, string): Outputs the given string at given world co-ordinate points.

polymarker (n, wcPoints) : It places the current marker symbol to all the n no. of points, whose co-ordinates are given in wcPoints. For eg. if marker is and there are 7 points specified in the wcPoints table, the output is as below:



Attributes: Line, Color, Area fill, Character

Line Attributes

- **Line Type**

setLinetype(It): It is used to set the type of line by passing the argument linetype - It, which has values 1,2,3 and 4 representing solid, dashed, dotted and dash-dotted respectively. Once the type of line is set by using this PHIGS function, all the line drawing command given afterwards, will draw selected type of line. Below example shows the plotting of some data using different line types.

- **Line width**

SetLinewidthScaleFactor (lw): The function sets the width of line by '1w' no. of times the original width. If value of w=1, the line is of standard width, if it is 0.5, the width is half of standard, above values greater than 1 produces thicker lines.

In raster scan, the lines of width> 1 are drawn by plotting the additional pixel which is adjacent to the line. If the width is 2, the line with (x,y) values is drawn with one more line with (x,y+1) values, If 1w is greater than 2, we select alternatively, the lines to right and left of the standard line.

Drawing thick lines, there is one problem that the horizontal and vertical lines can be perfectly drawn, but the diagonal lines will be drawn thinner (by factor lh/2).

Another problem is that the shape of line at the ends is horizontal or vertical. To solve this, we adjust the shape of ends b using the line caps. There are three types of line caps which can be set according to requirement, which are as below:

When two thick lines are intersecting, then also we may want to set the join into require shape. The three types of joins are as below:

Mitter join

A mitter join is accomplished by extending the outer boundaries of each of the two lines until they meet.

Round join

A round join is produced by capping the connection between the two segments with a circular boundary whose diameter is equal to the line width.

Bevel join

A bevel join is generated by displaying the line segments with butt caps and filling in the triangular gap where the segments meet.

If the angle between two connected line segments is very small, a mitter join can generate a long spike that distorts the appearance of the poly line.

- **Pen and Brush options**

With some packages, lines can be displayed with pen or brush selections. Options in this are shape, size and pattern. The different shapes for each pixel in line can be used. The size is used 'to set the width of such lines. The pattern is used to get different styles of brushing applied on the thick lines, like solid-brush, shadow type brush, etc. The 'shapes and patterns can be as below:

Lines generated with pen or brush shapes can be displayed in various widths by changing the size of the mask. for example,the rectangular pen line inFig. 4-9 could be narrowed width a 2*2 rectangular mask or widened with a 4*4 mask.

- **Line Color**

SetPolylineColourlndex (Ic) : The lines drawn after using this function contains the tine color as Ic.

polyline (n, wcpoints): This function is used to draw a polyline (more than 1 lines), where n is the total no. of tines, and the array wcpoints contains positions of endpoints of those lines.

Color and grayscale levels (Attributes)

Different color and intensity levels can be provided to the user, depending of the design of a particular system. In raster scan, there is wide range of colors available, but in random scan, there is few color choices, if any. The color options are numerically coded with the value ranging from 0 through the positive integers.

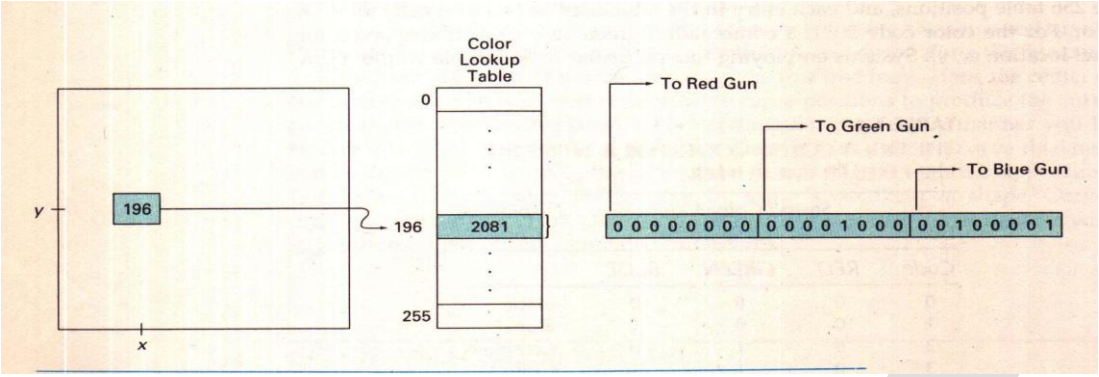
In color raster scan, no. of colors available depends on the storage capacity per pixel in the frame buffer. Minimum no. of colors can be provided by using 3 bits per pixel, each of which represents the presence of R, G and B, a frame buffer of which is as below.

The below table is the color table or a color lookup table or video lookup table.

Color	Stored Color Values in Frame Buffer			Displayed Color
Code	RED	GREEN	BLUE	
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

If the no. of bits per pixel is increased, the total no. of colors produced can be increased. Thus, if we use 24 bits per pixel, high resolution is obtained. But, the problem is that we require 3 MB memory storage for a frame buffer. The solution to this is that we can use the color look-up table or video look-up table. U only 8 bits per pixel in this table, we can obtain 7 million colors. Also, the storage requirement for frame buffer is reduced to 1 MB. The working is shown as below:

Figure shows that



The value at (x,y) is 196, and look-up table points to 2081, where the color information is stored with the 8 bits for each color, and accordingly, intensity of 3 colors will be set.

The PHIGS function to set the color table entries is given as **setColourRepresentation (ws, ci, colorptr)** : ws is workstation output device, ci is the color index and colorptr points to the values of RGB ranging from 0 to 1.

Grayscale

The monitors which do not have capability of producing colors, uses the different shades of gray, or grayscale for displaying the output primitive.

If we use 2 bits per pixel, there are four possible colors. Black,dark gray,light gray and white.

If 3 bits are used, we can get 8 shades of gray, and with 8 bits per pixel, there are 256 shades. The another method to store intensity information is to directly convert the intensity code into the voltage value which produces required grayscale level on the output device.

Area Fill Attribute

setinteriorStyle (fs). It is the PHIGS function to set interior style of a polygon area. The values of fs are hollow, solid, pattern and hatch. This function can also be applied to the area with curved boundaries. The color of the selected style is set by using set lnteriorColorlndex (fc): fc can be set to desired color code.

The hollow and solid styles in first function can be painted, but the pattern option requires that which pattern we want. For that the PHIGS function is given as **setInteriorStyleIndex (pi)**, where pi is the pattern index (the type of pattern required).

We can create our own pattern for the output device by using the function given by **setPatternRepresentation (ws, p1, nx, ny, cp)** : where ws is workstation (o/p device), pi is the pattern index which we want, cp is two dimensional array with nx rows and ny columns.

Thus, **setPatternRepresentation (1, 5, 2, 2, cp)** means that use device no. 1, create pattern no 5, whose pattern is in array cp with 2 rows and 2 columns, and this pattern will produce alternated red and white pixels. if we want more than one pixels for x and y, we can set by using:

setPatternSize (dx, dy), where dx and dy are wic:n and height of array mapping.

In filling the area with a pattern, we may want to start pattern with some specific point, known as tiling (like putting tiles from left top corner of room, or from centre). This can be done by function setPatternReferencePoint (position): position represents the x, y co-ordinates of lower left corner of a pattern.

We can also perform the Boolean operations on pattern as below:

Softfilling patterns: We may want to fill area again due to 2 reasons : it is blurred (unclear) when painted first time, or it is painted with semitransparent brush. Such algorithm is called Sofifill or tint-fill. If background color B, and foreground color F are not same, new color by combination

can be P = tF + (1-t)B, where t is called transparency factor. We can also have more than one backgound, and eqn. For that will be P = t_0 F+ t_1B_1 + (1- t_0 - t_1) B2.

Character Attributes

- **setTextFont (tt)** The paramter tf is the text font given by the integer value. The desig style may be different such as NewYork, Courier, TimesRoman etc., with the underlining styles solid dotted and double which may be boldface, italic, : or shadow.

- **set TextColourindex (tc) :** Here the tc is the text color which is required. By using this function, each pixel of the character design will be painted with the color integer value of parameter tc.

- **setCharacterHeight (ch):** It is used for scaling cf the height of the character font.

- **setCharacterExpansionFactor (cw):** It is used to set the width of a character.

- **setCharacterSpacing (Cs):** The parameter ch represents character spacing which may be any real value. It is used to set the space between two characters.
  Eg. SpacingO S p a c i n g 0.5 S p a c i n g I

- **set TextPath (tp)**: It can have four values : left, right, up or down.

- **setCharacterup Vector (upvect)** : The upvect is the function which has two values of orientation in x and y co-ordinate. If upvector (1,1) it is 45°, as below:

- **set TextAlignment (h, v)**: It is used to specify horizontal and vertical alignment of text.

- **setTextPrecision (tpr)** : The precision of text has values string, char or stroke. The stroke is having highest precision. Thus, according our requirement of precision, the value is set.