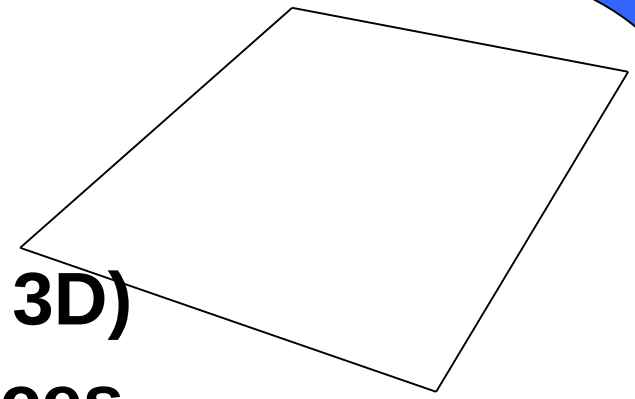
Filled Area Primitives

By
Poonam Saini
Dept. of Computer Science & Engineering
Sir Padampat Singhania University
Udaipur

Filled Area Primitives

Polygon



- Vertex = point in space (2D or 3D)
- Polygon = ordered list of vertices

→Each vertex connected with the next in the list

→Last is connected with the first

→Maybe more than one – polygons with holes

→May contain self-intersections

- A surface which is closed one and bounded by a straight line segments is known as polygon.
- The line segments are called edges or sides of the polygon.
- The point of intersection of two edges is called as vertex of the polygon.

Polygon

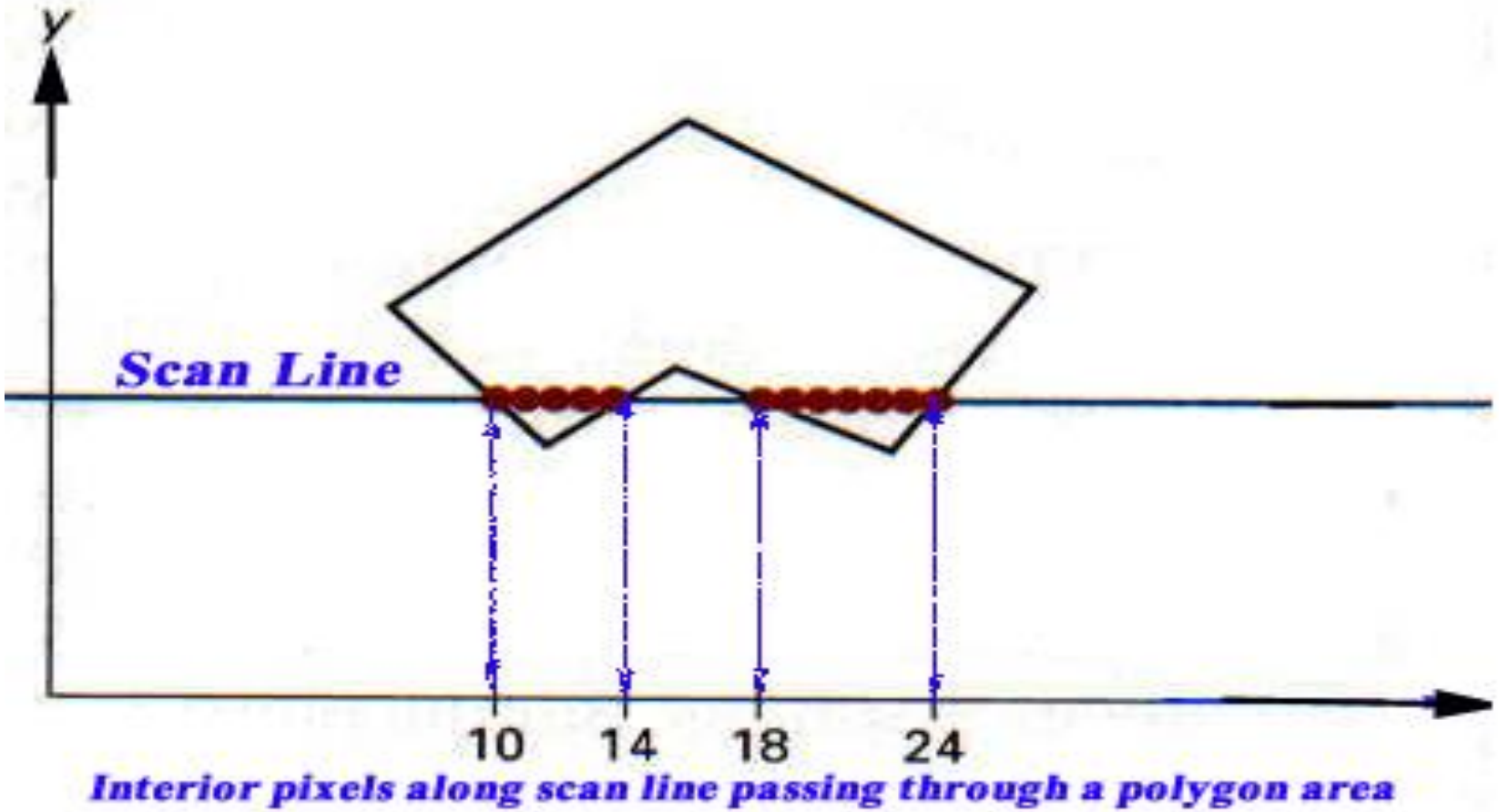
- Types of Polygons:

1. **Regular Polygons:** The polygons having equal length of all edges with internal/external angle between any two connected edges are same are known as regular polygons.
2. **Irregular Polygons:** The polygons having different length of edges and the angles between any two connected edges are same.
3. **Convex Polygons:** Take two points inside the polygon and join these points by a straight line. If all the points on this line lies inside the polygon, then its is a convex polygon.
4. **Concave Polygon**

Scan-Line Polygon Fill Algorithm

- 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.

Scan-Line Fill Algorithm



Interior pixels along scan line passing through a polygon area

Scan-Line Polygon Fill Algorithm

- Calculations performed in scan-conversion and other graphics algorithms typically take advantage of various coherence properties of a scene that is to be displayed.
- **Coherence** is simply that the properties of one part of a scene are related in some way to other parts of the scene so that the relationship can be used to reduce processing.
- Coherence methods often involve incremental calculations applied along a single scan line or between successive scan lines.

Scan-Line Polygon Fill Algorithm

- **Spatial Coherence:** The adjacent pixels are likely to have the same characteristics. This property is known as spatial coherence.

- **Scan Line Coherence:** Adjacent pixels on a scan line are likely to have the same characteristics. This is called scan line coherence.

Note: These two properties are useful in scan converting a polygon.

Scan-Line Polygon Fill Algorithm

- Some scan line intersections at polygon vertices require special handling.
- For eg. : A scan line passing through a vertex intersects two polygon edges at that position

Scan-Line Polygon Fill Algorithm

- Scan line y' generates an even number of intersections that can be paired to identify correctly the interior pixel spans.
- To identify the interior pixels for scan line y, we must count the vertex intersection as only one point.

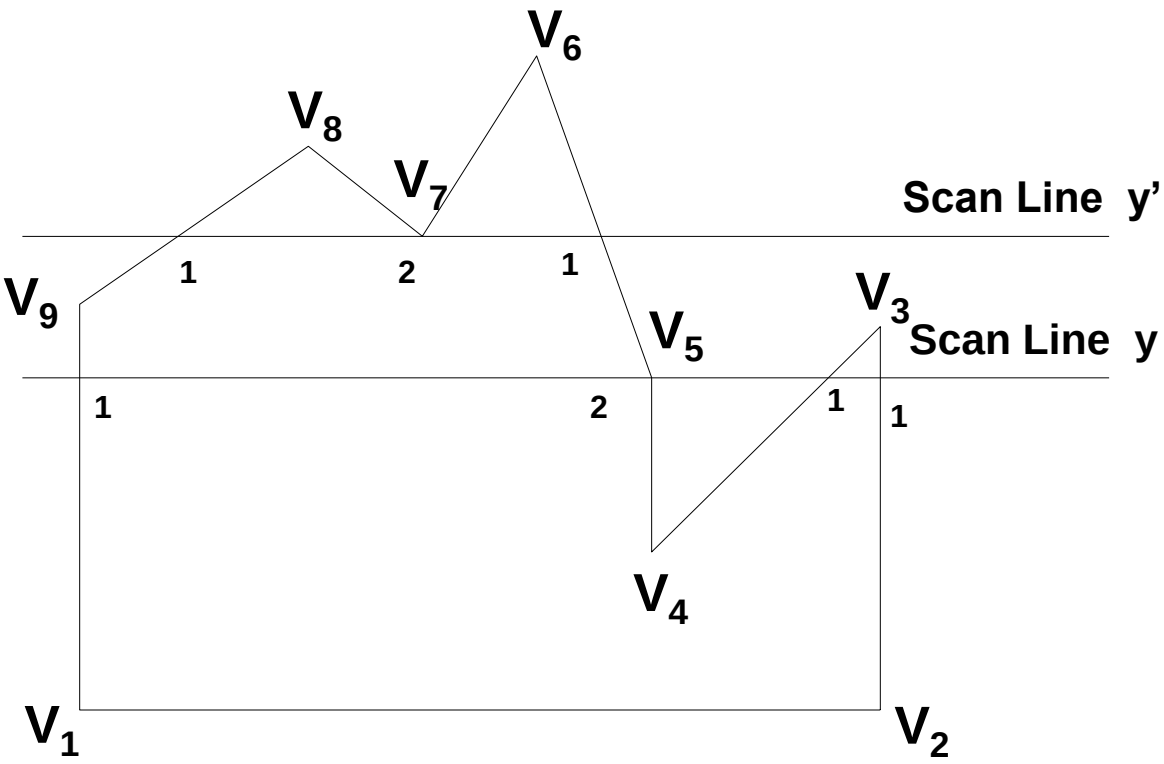


Fig.: Intersection points along the scan lines that intersect polygon vertices.

Scan-Line Polygon Fill Algorithm

- In case of scan line y, we need to do some additional processing to determine the correct interior points.
- Consider the vertex V_7, both the edges are on the same sides whereas in case of vertex V_5 both the edges are in the opposite direction. Therefore we have to work on V_5.
- These kind of vertices can be traces either in clockwise direction or anticlockwise and observe relative changes in y coordinates of the vertex.

Scan-Line Polygon Fill Algorithm

- If the end point y values of two consecutive edges monotonically increases or decreases, we count the middle vertex as a single intersection point for any scan line passing through that vertex otherwise two edge intersections can be added to the intersection list.

Scan-Line Polygon Fill Algorithm

- **In determining the edge intersection, we can set up an incremental coordinate calculation along any edge by exploiting the fact that the slope of the edge is constant from one scan line to the next.**

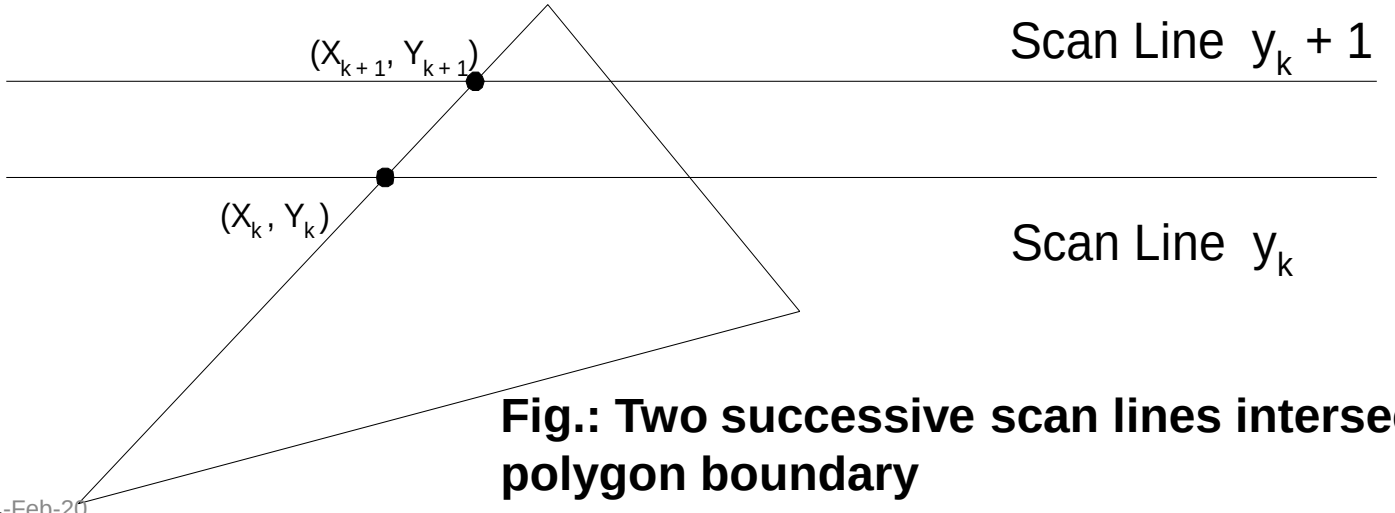


Fig.: Two successive scan lines intersecting polygon boundary

Scan-Line Polygon Fill Algorithm

- The slope of the edge is constant from one scan line to the next:
  - let m denote the slope of the edge.

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

$$x_{k+1} = x_k + \frac{1}{m}$$

- Each successive x is computed by adding the inverse of the slope and rounding to the nearest integer

Scan-Line Polygon Fill Algorithm

- **Recall that slope is the ratio of two integers:**

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

- **So, incremental calculation of x can be expressed as**

$$x_{k+1} = x_k + \frac{\Delta y}{\Delta x}$$

Inside-Outside Tests

- Area-filling algorithms and other graphics processes often need to identify interior regions of objects.

- To identify interior regions of an object graphics packages normally use Odd-Even method for inside-outside test.

Inside-Outside Tests

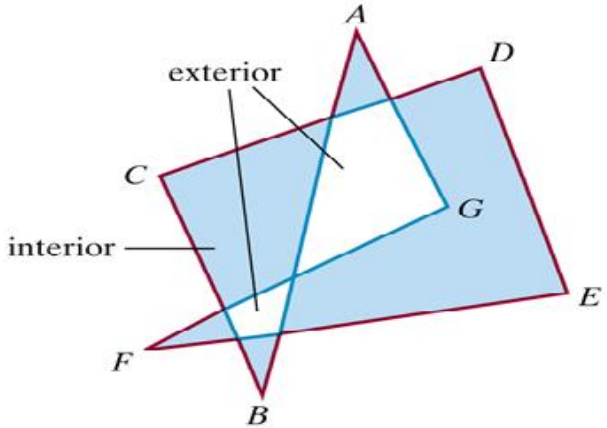
Odd-Even rule (Odd Parity Rule, Even-Odd Rule):

1. draw a line from any position P to a distant point outside the coordinate extents of the object and counting the number of edge crossings along the line.
2. If the number of polygon edges crossed by this line is odd then

   P is an interior point.

   <u>else</u>

   P is an exterior point



Odd-Even Rule

Sorted Edge Table

- **In SET, there is an entry for each scanline.**
- **Traverse edges of the polygon to construct a Sorted Edge Table (SET)**

  1.Eliminate horizontal edges

  2.Add edge to linked-list for the scan line corresponding to the y_lower vertex. Shorten edges if necessary to resolve the vertex-intersection problem.

  3.For each edge entry, store the following:

  - **y_upper: the largest y value on that edge (last scanline to consider)**
  - **x_lower: the x intercept at that scanline (initial x value)**
  - **1/m: for incrementing x**

  4.For each scan line the edges are sorted from left to right (based on x)

Active Edge Table

- Construct Active Edge List during scan conversion. AEL is a linked list of active edges on the current scanline, y. The active edges are kept sorted by x

  →The active edge list contains all the edges crossed by that scan line.

  →As we move up, update the active edge list using the sorted edge table if necessary.

Scan line polygon fill Algorithm

1.Set y to the smallest y coordinate that has an entry in the SET; i.e, y for the first nonempty bucket.

2.Initialize the AEL to be empty.

3.For each scanline y repeat:

(a) Copy from SET bucket y to the AEL those edges whose y_min = y (entering edges).

(b) The sort the AEL on x is easier because SET is presorted.

(c) Fill in desired pixel values on scanline y by using pairs of x coordinates from AEL.

(d) Remove from the AEL those entries for which y = y_max (edges not involved in the next scanline.)

(e) Increment y by 1 (to the coordinate of the next scanline).

(f) For each nonvertical edge remaining in the AEL, update x for the new y.

Boundary-Fill Algorithm

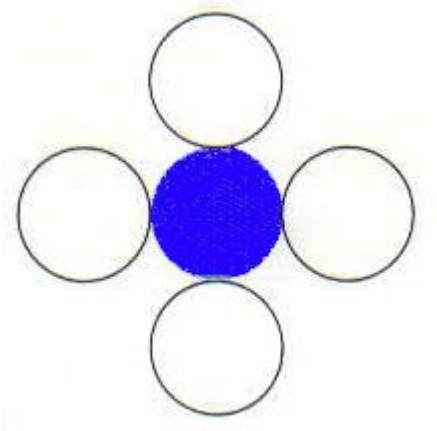
- Start at a point inside a region and paint the interior outward toward the boundary. If the boundary is specified in a single color, the fill algorithm proceeds outward pixel by pixel until the boundary color is encountered.
- It is useful in interactive painting packages, where interior points are easily selected.
- The inputs of the this algorithm are:
  - Coordinates of the interior point (x, y)
  - Fill Color
  - Boundary Color

Boundary-Fill Algorithm (contd..)

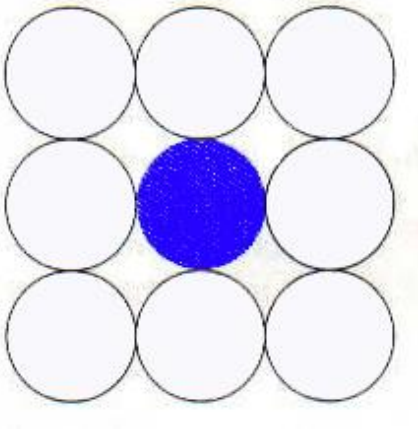
- Starting from (x, y), the algorithm tests neighboring pixels to determine whether they are of the boundary color. If not, they are painted with the fill color, and their neighbors are tested. This process continues until all pixels up to the boundary have been tested.
- There are two methods for proceeding to neighboring pixels from the current test position:

Boundary-Fill Algorithm (contd..)

1. The 4-connected method.



2. The 8-connected method.

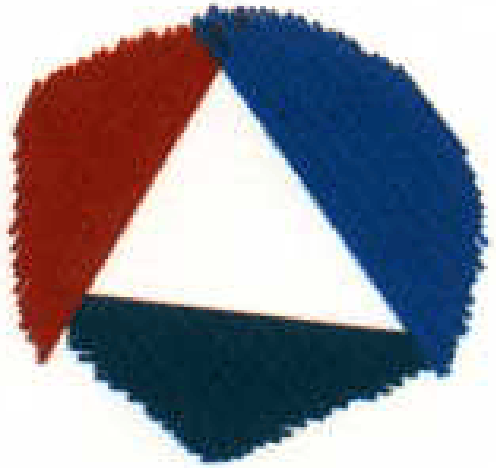


Boundary-Fill Algorithm (contd..)

```
void boundaryFill4 (int x, int y, int fillColor, int borderColor)
{ int interiorColor;
/* set current color to fillColor, then perform following operations. */
  getPixel (x, y, interiorColor);
  if ( (interiorColor != borderColor)  &&  (interiorColor != fillColor) )
     {
        setPixel (x, y);   // set color of pixel to fillColor
        boundaryFill4  (x + 1, y, fillColor, borderColor);
        boundaryFill4 (x - 1,   y, fillColor, borderColor);
        boundaryFill4 (x, y + 1,  fillColor, borderColor);
        boundaryFill4 (x, y - 1,   fillColor, borderColor);
} }
```

Flood-Fill Algorithm

- Sometimes we want to fill in (or recolor) an area that is not defined within a single color boundary. We can paint such areas by replacing a specified interior color instead of searching for *a* boundary color value. This approach is called a flood-fill algorithm.



Flood-Fill Algorithm

- We start from a specified interior point (*x, y*) called the seed point . Then using a 4-connected or 8-connected approach, the entire area can be filled by a specified color.
- If they have a color which is that of the boundary then do not consider the pixel.
- If color is different then color the pixel with the desired color.
- The process will be recursive one and stopped when there is no neighbouring pixel which can be colored.

Flood-Fill Algorithm

```
void floodFill4 (int x, int y, int fillColor, int interiorColor)
{ int color;
/* set current color to fillColor, then perform following operations. */
  getPixel (x, y, color);
  if  (color = interiorColor)
  {
        setPixel (x, y);   // set color of pixel to fillColor
        floodFill4  (x + 1, y, fillColor, interiorColor);
        floodFill4 (x - 1,   y, fillColor, interiorColor);
        floodFill4 (x, y + 1,  fillColor, interiorColor);
        floodFill4 (x, y - 1,   fillColor, interiorColor);
  }
}
```

Attributes of Output Primitives

Attributes of Output Primitives

- Definition: Parameter that affects the way a primitive will be displayed.
- Consider only those attributes that control the basic display properties of primitives.
- For eg.: Lines can be dotted or dashed. Fat or thin & blue or orange.

Line Attributes

- Basic attributes of a Line are
    1. Type
    2. Width
    3. Color

Line Attributes

- **1. Type Attribute**
- **Solid** _____
- **Dotted – very short dash with spacing equal to or greater than dash itself.**

.....

- **Dashed – displayed by generating an interdash spacing**

- -

Line Attributes

- **2. Width Attribute**
- **Specify in pixels and proportion of a standard line width.**
- **Thicker line can be produced by:**
  - **Adding extra pixel vertically when |m| < 1**
  - **Adding extra pixel horizontally when |m| > 1**
- **Issues:**
  - **Line have different thickness on the slope.**
  - **Problem with End of the line and Joining the two lines (polygon)**

Line Attributes

- **2. Width Attribute**

The problem raised due to different shapes at the end of the line can be solved by adding line endcaps.

Line Attributes

- **2. Width Attribute**

(a) Butt Cap: It is obtained by adjusting the end positions of the lines so that the thick line is displayed with square ends are perpendicular to the line path. If specified line has slope m, the square end of the thick line has slope -1/m.

Butt Cap

Line Attributes

2. Width Attribute

(b) Round Cap: It is obtained by adding a filled semicircle to each butt cap. The circular areas are centered on the line endpoints and have a diameter equal to the line thickness.

Round Cap

Line Attributes

- **2. Width Attribute**

(c) Projecting Square Cap: In this type, we simply extend the line and add butt caps that are positioned one half of the line width beyond the specified end points.

Projecting
Square Cap

Line Attributes

- **2. Width Attribute**

A smoothly connected series of line segments can not be produced as thick lines using horizontal and vertical pixel spans, leave pixel gaps at the boundaries between lines of different slopes where there is a shift from horizontal spans to vertical spans. There are some method for smoothly joining two lines of segments.

Line Attributes

Methods for smoothly joining two lines of segments

1. **Mitter Join : It is accomplished by extending the outer boundaries of each of the two lines until they meet.**

Miter Join

Line Attributes

Methods for smoothly joining two lines of segments

2. Round Join : It is produced by capping the connection between the two segments with a circular boundary whose diameter is equal to the line width.

Round Join

Line Attributes

Methods for smoothly joining two lines of segments

3. Bevel Join : It is generated by displaying the line segments with butt caps felling in triangular gaps where the segments meet.

Bevel Join

Line Attributes

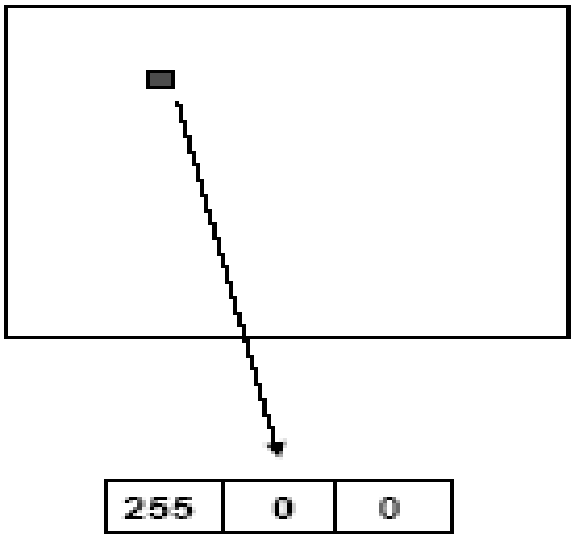
3. Color Attribute

- Colors are represented by colors codes which are positive integers.
- Color information is stored in frame buffer or in separate table and use pixel values as index to the color table.
- Number of color choices depends on the no. of bits available per pixel in the frame buffer.
- Two ways to store color information :
    1. Direct
    2. Indirect

Line Attributes

- 1. Direct

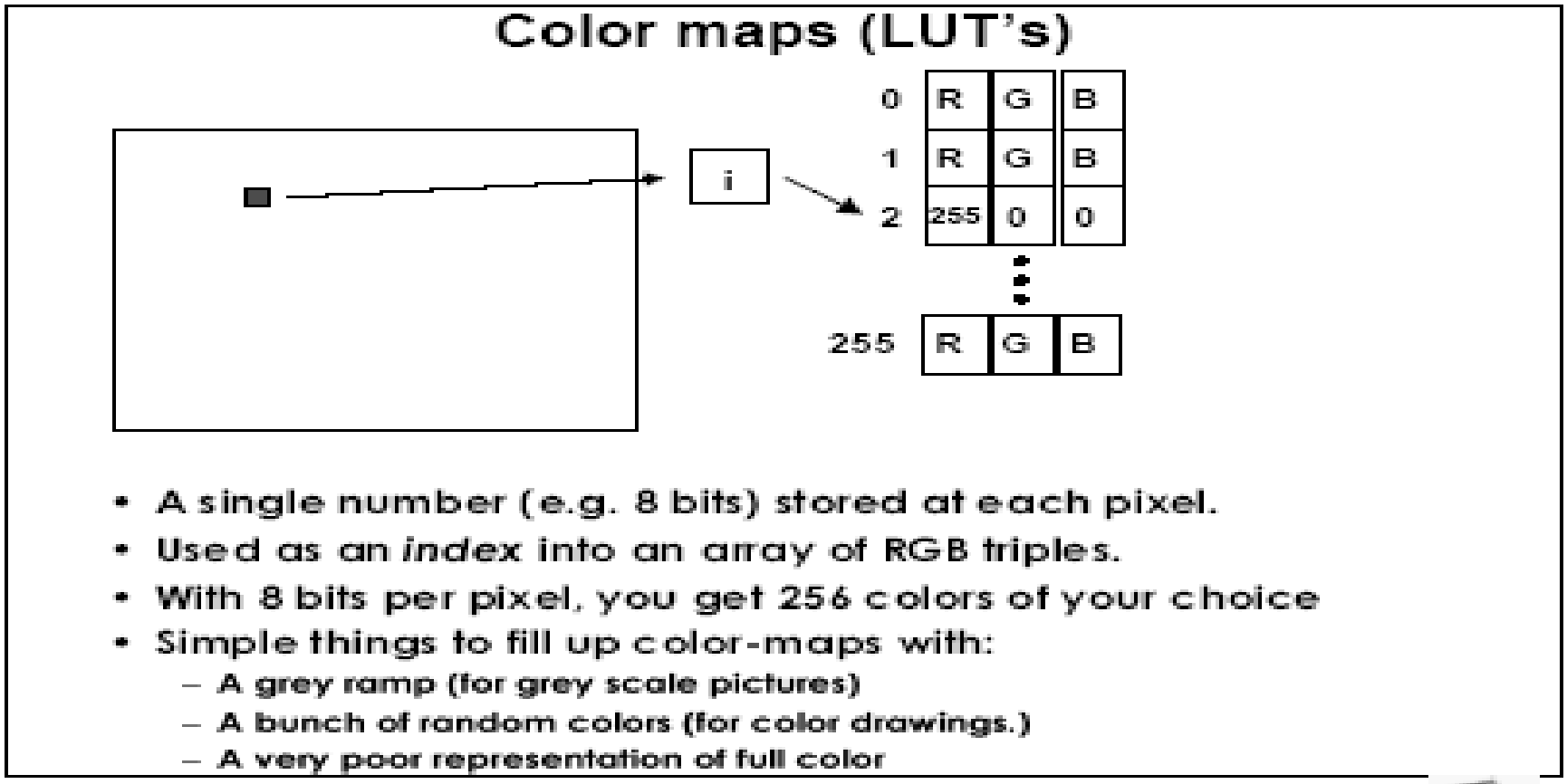
Full-color (RGB) displays



- For 24 bit color:
  - store 8 bits each of red, green, and blue per pixel.
  - E.g. (255,0,0) is pure red, and (255, 255, 255) is white.
  - Yields 2^24 = 16 million colors.
- For 15 bit color:
  - 5 bits red + 5 green + 5 blue
- The video hardware uses the values to drive the R,G, and B guns.
- You can mix different levels of R, G, and B to get any color you want

Line Attributes

- 1. Indirect



Character Attributes

- The appearance of displayed characters is controlled by attributes such as font, size, color, and orientation.

- Attributes can be set both for entire character strings (text) and for individual characters defined as marker symbols.

Character Attributes

1. Text Attributes:

- First of all, there is the choice of font (or typeface), which is a set of characters with a particular design style such as New York, Courier, Helvetica, London, 'Times Roman', and various special symbol groups.
- The characters in a selected font can also be displayed with assorted underlining styles (solid, dotted , double), in boldface, in *italics*. and in outline or shadow styles.

Character Attributes

1. Text Attributes:

- A particular font and associated style is selected in a PHIGS program by setting an integer code for the text font parameter tf in the function.
- Color settings for displayed text are stored in the system attribute list and used by the procedures that load character definitions into the frame buffer.
- When a character string is to be displayed, the current color is used to set pixel values in the frame buffer corresponding to the character shapes and positions

Character Attributes

1. Text Attributes:

- **Point measurements specify the size of the body of a character but different fonts with the same points specifications can have different character size depending on the design of the typeface.**

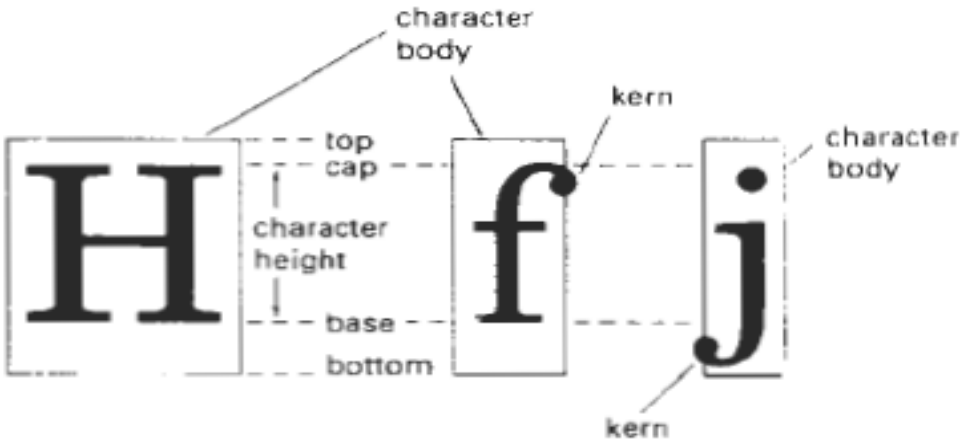


Figure 4-23
Character body.

Character Attributes

1. Text Attributes:

- The distance between the bottom line and the top line of the character body is the same for all characters in a particular size and typeface, but the body width may vary.
- Character height is defined as the distance between the baseline and the cap line of characters.

Character Attributes

Orientation

The orientation for a displayed character string is set according to the direction of the character up vector in which x and y components are specified.

- Text is then displayed so that the orientation of characters from baseline to capline is in the direction of the up vector.

Character Attributes

Orientation

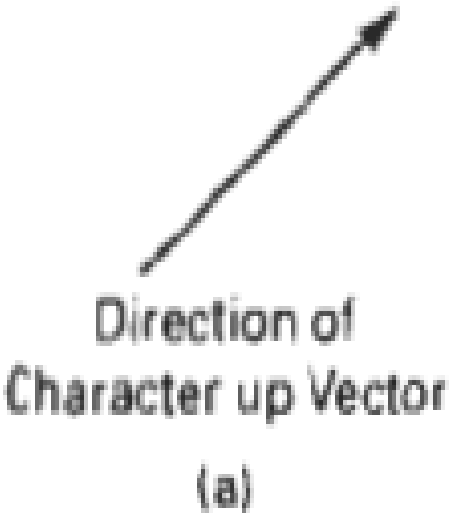
→The direction of the up vector is $45^0$and text would be displayed as

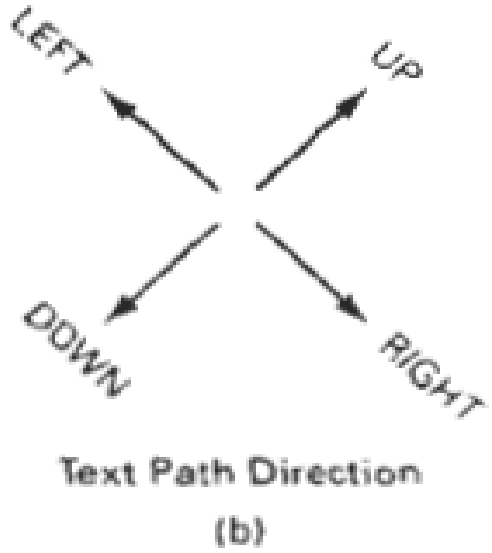
Direction of the up vector (a) controls the orientation of displayed text (b).

Character Attributes

Orientation

→ **A combination of textpath and upvector can be used to display the slanting text. As Up vector controls the direction of the text path.**





Character Attributes

Orientation

→Text-path can be set in right, left, up or down

Text displayed with the four text-path options.

Character Attributes

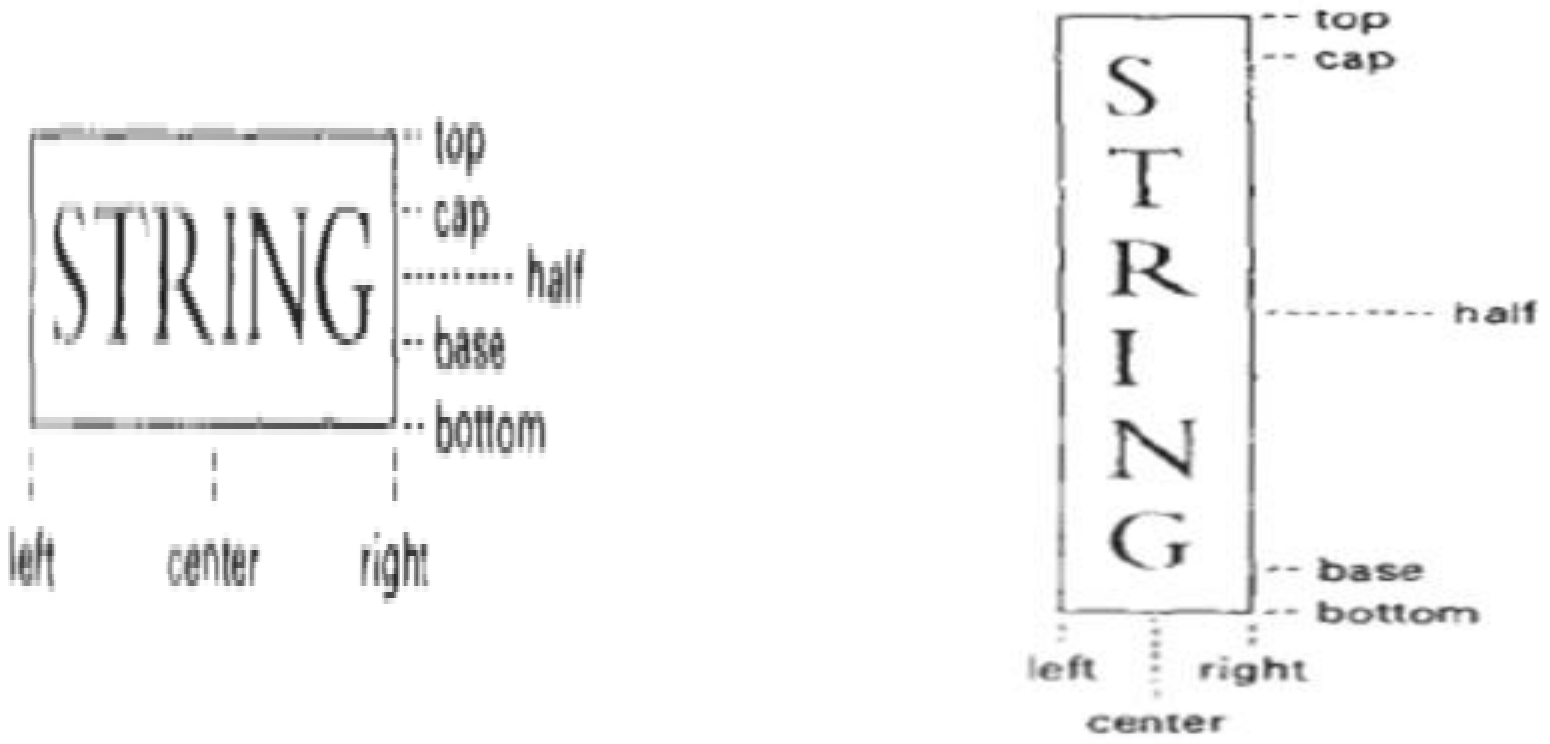
Text Alignment

→This attribute specifies how text is to be positioned with respect to the start coordinates.

- Horizontal alignment is set by assigning value left, centre, or right.
- Vertical alignment is set by assigning value top, cap, half, base, or bottom

Character Attributes

Text Alignment



Alignment attribute values for horizontal and vertical strings.

Area-Fill Attributes

- Options for filling a defined region include a choice between a solid color or a patterned fill and choices for the particular colors and patterns. These fill options can be applied to polygon regions or to areas defined with curved boundaries, depending on the capabilities of the available package. In addition, areas can be painted using various brush styles, colors, and transparency parameters.

Area-Fill Attributes

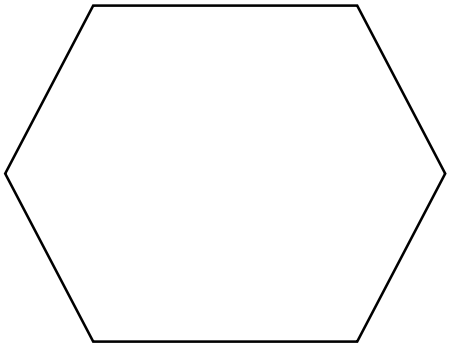
Fill Styles:

- Areas are displayed with three basic fill styles:

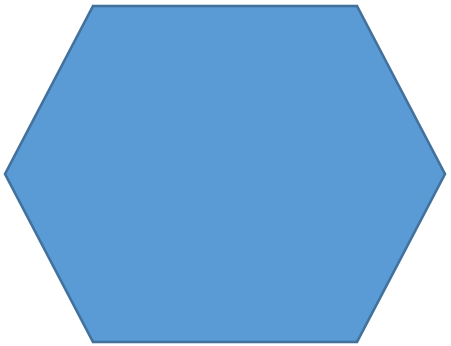
  -Hollow with a color border

- filled with a solid color,

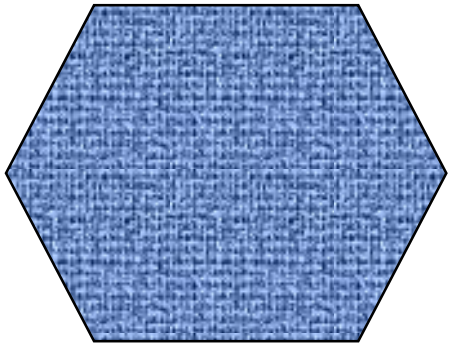
- filled with a specified pattern or design.



Hollow



Solid



Patterened

Area-Fill Attributes

Fill Styles:

- A polygon hollow fill is generated with line drawing routines as a closed polyline.
- Another value for fill style is hatch, which is used to fill an area with selected hatching patterns-parallel lines or crossed lines.

Polygon fill using hatch patterns.

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

Area-Fill Attributes

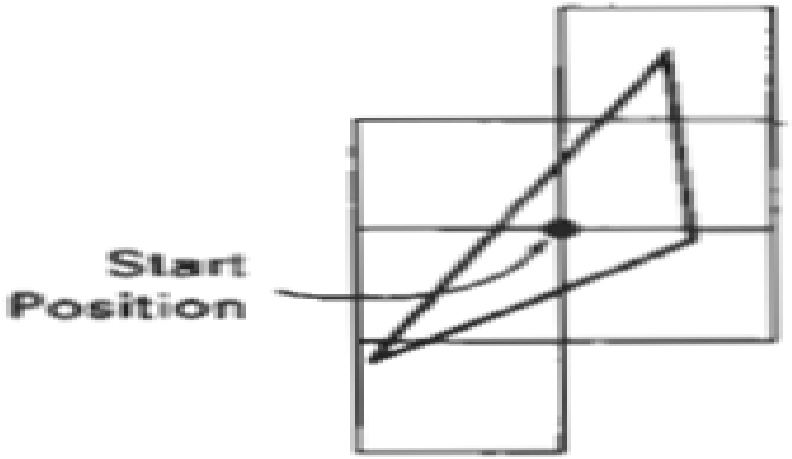
Fill Styles:

- Solid fill of a region can be accomplished with the scan-line procedures.
- Other fill options include specifications for the edge type, edge width, and edge color of a region.
- These attributes are set independently of the fill style or fill color, and they provide for the same options as the line-attribute parameters

Area-Fill Attributes

Pattern Fill:

- The process of filling an area with a rectangular pattern is called tilling and rectangular fill patterns are refered to as tiling pattern.



Tiling an area from a designated start position. Nonoverlapping adjacent patterns are laid out to cover all scan lines passing through the defined area.

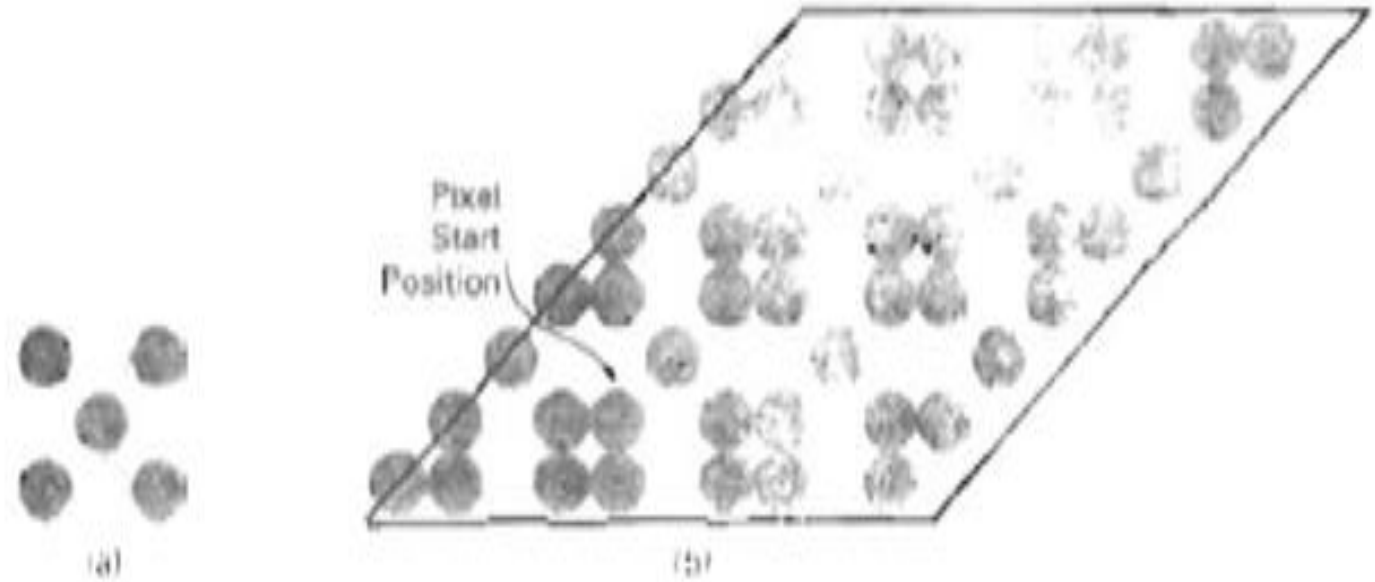
Area-Fill Attributes

Pattern Fill:

- To superimpose a selected pattern on scan lines, the scan line procedures are to be modified.
- Beginning from a specified start position for pattern fill, the rectangular patterns should be mapped vertically to scan lines between the top and bottom of fill area and horizontally to interior pixel position across these scan lines.

Area-Fill Attributes

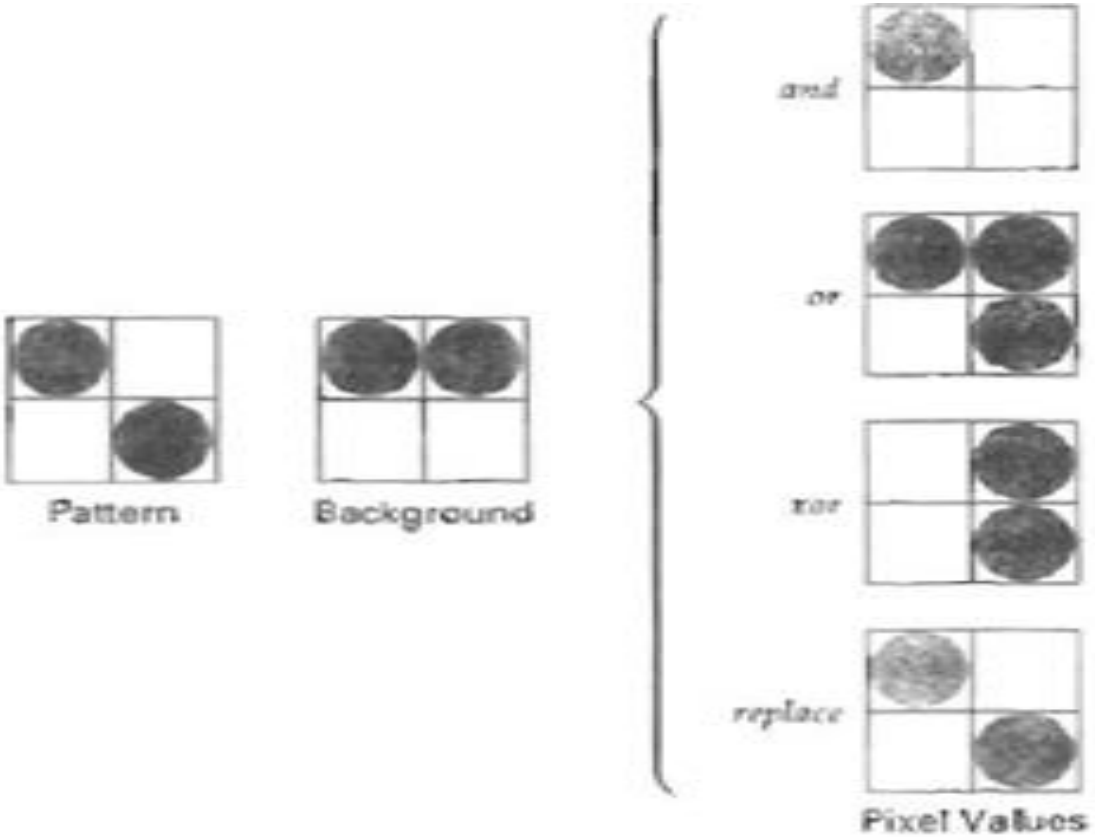
Pattern Fill:



A pattern array (a) superimposed on a parallelogram fill area to produce the display (b).

Area-Fill Attributes

Pattern Fill:



Combining a fill pattern with a background pattern using Boolean operations, *and*, *or*, and *xor* (*exclusive or*), and using simple replacement.

Area-Fill Attributes

Soft Fill:

- Modified boundary-fill and flood-fill procedures that are applied to repaint areas so that the fill color is combined with the background colors are referred to as soft-fill .
- One use for these fill methods is to soften the fill colors at object borders that have been blurred to antialias the edges.
- Another is to allow repainting of a color area that was originally filled with a semitransparent brush, where the current color is then a mixture of the brush color and the background colors "behind" the area. In either case, we want the new fill color to have the same variations over the area as the current fill color.