Subject :→ Computer Graphics.

Experiment . No. – 5

Aim :→ To implement area filling algorithms :→ Boundary fill , Flood Fill.

Theory :→

Filled Area Primitive :→

In computer graphics, a Filled Area Primitive refers to a basic graphical element used to represent filled areas, such as polygons or other shapes, with a color, texture, or pattern. These primitives are used to fill enclosed regions, typically bounded by a set of edges, and are a core concept in rendering complex shapes. The most common filled area primitives include polygons, rectangles, circle and ellipses or other arbitrary regions.

Area Filling Algorithms :→

An area filling algorithm is a technique used in computer graphics to fill a bounded region (area) with a particular color or pattern. These algorithms are widely used in painting programs, image editing software, and in rendering shapes in graphics applications. The primary goal of such algorithms is to fill a connected region inside a boundary.

Here are the two main types of area filling algorithms :→ 1] Seed Fill Algorithms

- Boundary Fill Algorithm
- Flood Fill Algorithm

2] Scan Fill Algorithm.

(vi) Sometime 4-connectivity fails to paint the entire region. 8-connectivity is time consuming and memory intensive.

(vii) This method is useful in interactive painting packages, where the selection of interior pixel can be done very easily using input device like a mouse.

(viii) To create a solid region, set the fill color to boundary color, so that boundary and interior region becomes in distinguish able after filling.

(ix) Recursively this algorithm checks all pixels in given polygon and fills them if not already filled.

Algorithm :→

```
BOUNDARY_FILL (x, y, fillColor, boundaryColor)
    Current ← getcolor(x,y)
    if current ≠ fillColor && current ≠ boundaryColor
    then, putPixel(x, y, fillColor)
4-way  { BOUNDARY_FILL (x+1, y, fillcolor, boundaryColor)
connectivity  BOUNDARY_FILL (x-1, y, fillcolor, boundaryColor)
    BOUNDARY_FILL (x, y+1, fillcolor, boundaryColor)
    BOUNDARY_FILL (x, y-1, fillColor, boundaryColor)
8-way connectivity  BOUNDARY_FILL (x-1, y-1, fillcolor, boundaryColor)
    BOUNDARY_FILL (x-1, y+1, fillcolor, boundaryColor)
    BOUNDARY_FILL (x+1, y-1, fillcolor, boundaryColor)
    BOUNDARY_FILL (x+1, y+1, fillcolor, boundaryColor)
END
```

Seed Fill Algorithms :→

The seed Filling Algorithm is an area-filling algorithm used in computer graphics to fill a contiguous region in a graphical image or grid. It starts from a given seed point and spreads outward, filling neighboring pixels that match certain criteria (e.g., color or boundary conditions). The two main types of seed filling algorithms are Boundary Fill Algorithms and Flood Fill Algorithm.

Types of Seed Filling Algorithms :→

1. Boundary Fill Algorithms :→ (Edge Fill Algorithms)

Working Mechanism :→

(i) Boundary fill algorithm starts with some interior pixel of a polygon, called seed pixel and keep filling neighbor pixels in outward direction until the boundary color is encountered.

(ii) Boundary fill algorithm starts with three parameters :→ interior point (x, y), fill color and boundary color.

(iii) This approach retrieve the color of the current pixel and compares it with fill color and boundary color.

(iv) If the color of the current pixel is neither fill color nor boundary color, then fill it with the fill color and make a recursive call, otherwise skip the pixel under consideration.

(v) Neighbour pixel are approached using 4-connectivity or 8-connectivity.

Advantages →
(i) Simple
(ii) Easy to implement
(iii) Boundary fill algorithm cannot handle the objects with multi-color boundary, whereas flood fill can.

Disadvantages →
(i) Require extensive stacking.
(ii) Slow in nature
(iii) Not suitable for large polygon.

Conclusion →
The experiment aimed to implement Boundary Fill and Flood Fill algorithms for area filling in Computer graphics.
- Boundary fill algorithm → Worked well for regions with clearly defined boundaries but struggled with open or incomplete boundaries. It also faced stack overflow issues in large areas due to recursion.
- Flood Fill algorithm → Effectively filled contiguous regions of uniform color but was sensitive to slight color variations. Similar to Boundary Fill, it suffered from performance issues with large areas when using recursion.

Both algorithms are effective for specific use cases but require iterative implementation for large scale region to avoid recursion problems.

2) Flood Fill Algorithm :→

Working Mechanism :→

(i) Many realistic objects have different colored boundaries. Boundary fill algorithm cannot fill the polygon with multi-colored boundary.
(ii) Flood fill algorithm can handle this issue.
(iii) The algorithm starts with the interior pixel, fill color and old color. If the color of the current pixel is the same as old color then it fills ~~algorithm~~ the pixel with fill color and examines its neighbors recursively.
(iv) Like boundary fill algorithm, if the interior pixel is already filled with some other color, flood fill algorithm fails.

To handle this problem, we shall first point all interior pixel with background (old) color.

~~Algo~~ Algorithm :→

```
Flood_Fill (x, y, fill color, old Color)
  Current ← getColor (x, y)
  if Current ≠ old Color then
  put pixel (x, y, fillcolor)
  Flood_Fill (x+1, y, fill color, old color)      } 4-way Connectivity
  Flood_Fill (x-1, y, fill color, old color)
  Flood_Fill (x, y+1, fill color, old color)
  Flood_Fill (x, y-1, fillColor, old Color)
  Flood_Fill (x+1, y-1, fillcolor, old Color)     } 8-way Connectivity
  Flood_Fill (x+1, y+1, fill color, old Color)
  Flood_Fill (x-1, y-1, fillColor, old Color)
  Flood_Fill (x-1, y+1, fill color, old color)
END
```

Note :→

(i) This algorithm may not work properly if some of the pixels are already filled, because algorithm returns when the currents pixel is either boundary pixel or it is filled.

(ii) To avoid this, we shall set the color of all interior pixel to background color.

Advantages :→

(i) Simple
(ii) Easy to implement.
(iii) Works for any type of polygons.

Disadvantages :→

(i) Require extensive stacking due to recursion.
(ii) Does not work if the boundary is specified with multiple colors.
(iii) Need to specify seed point contained within the polygon.
(iv) Polygon should be filled with the background color.
(v) Require more memory.
(vi) Not suitable for the large polygon.