Computer Graphics

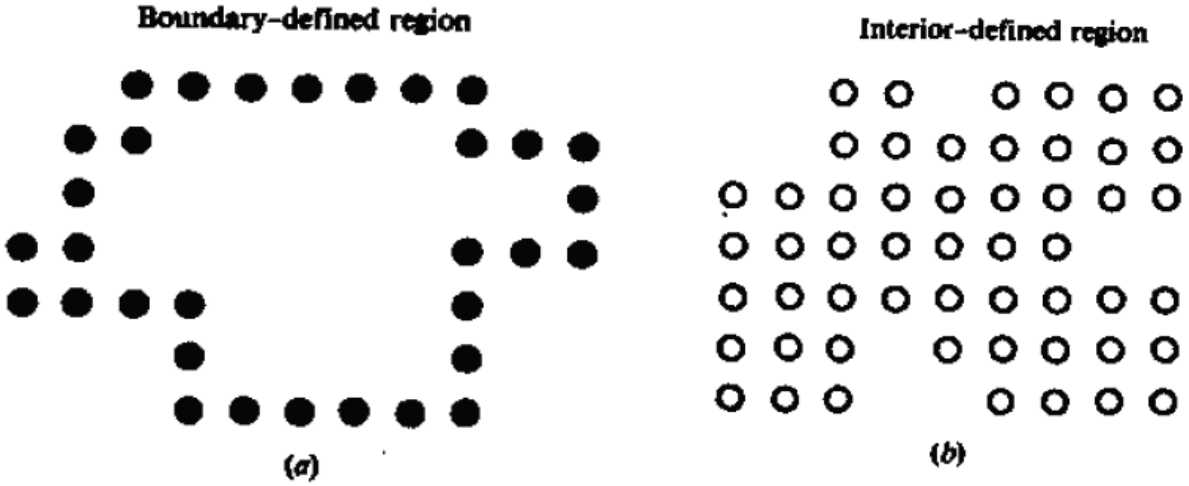
Lecture-04
Scan Conversion

Md Imtiaz Ahmed

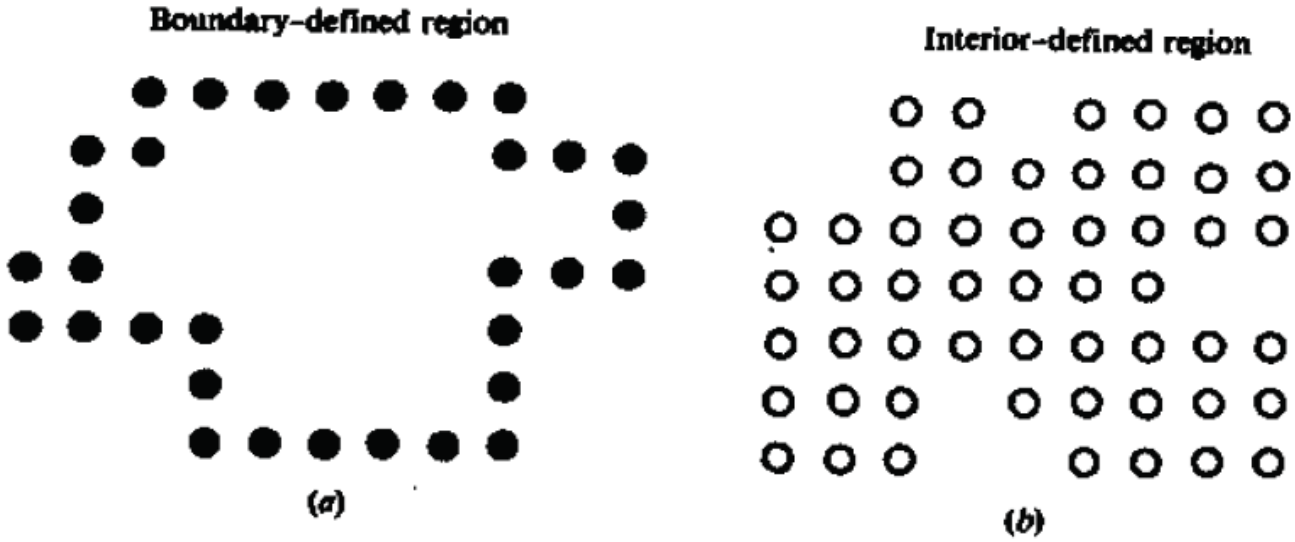
Lecturer, DIIT

Region Filling

- Region filling is the process of “coloring in” a definite image area or region.
- defined at the pixel or geometric level.
- At the pixel level, we describe a region either in terms
  - of the bounding pixels that outline it
  - or as the totality of pixels that comprise it



Region Filling



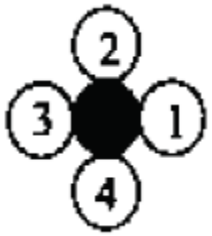
- In the first case the region is called boundary-defined region and the collection of algorithms used for filling such a region are collectively called boundary-fill algorithms.
- The other type of region is called an interior-defined region and the accompanying algorithms are called flood-fill algorithms.

Region Filling

- At the geometric level a region is defined or enclosed by such abstract contouring elements as connected lines and curves.
- For example, a polygonal* region, or a filled polygon, is defined by a closed polyline, which is a polyline (i.e., a series of sequentially connected lines) that has the end of file last line connected to file beginning of file first line.

4-Connected vs. 8-Connected

- There are two ways in which pixels are considered connected to each other to form a "continuous" boundary.
- One method is called 4-connected, where a pixel may
  - have up to four neighbors [see Fig. **3-10(a)];**

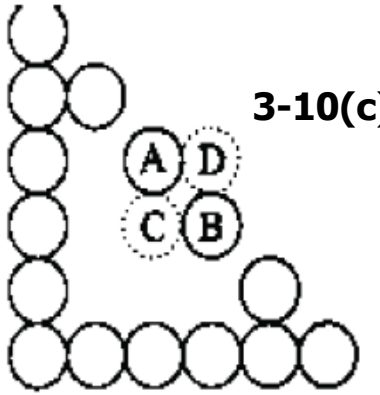


3-10(a)

- **the other is called 8-connectecd where a pixel may**
  - have up to eight neighbors [see Fig. **3-10(b)].**

3-10(b)

- **Using the 4-connected approach, the pixels in Fig.3-10(c) do not define a region since several pixels such as A and B are not connected.**
- **using the 8-connected definition we identify a triangular region.**



3-10(c)

Boundary-Fill Algorithm

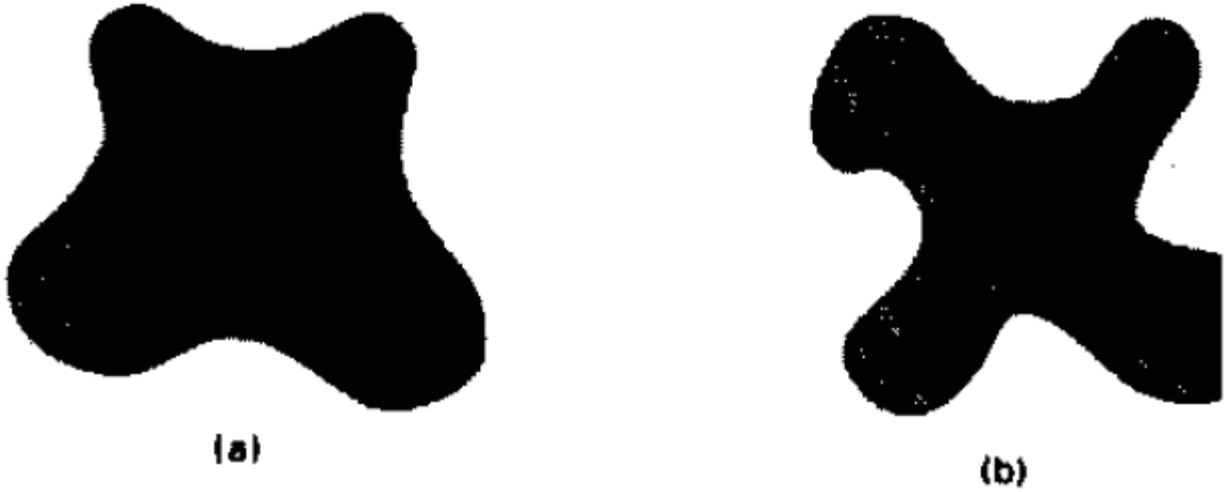
- Recursive algorithm that begins with a starting pixel, called a seed inside the region.
- The algorithm checks to see if this pixel is a boundary pixel or has already been filled.
- If the answer is no, it fills the pixel and makes a recursive call to itself using each and every neighboring pixel as a new seed.
- If the answer is yes, the algorithm simply returns to its caller.

Boundary-Fill Algorithm

- A boundary-fill procedure accepts as input the coordinates of an interior point (x, y), a fill color, and a boundary color.
- Starting from (x, y), the procedure tests neighboring positions 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 color for the area have been tested.
- Both inner and outer boundaries can be set up to specify an area.

Boundary-Fill Algorithm

- Example color boundaries for a boundary-fill procedure:



Boundary-Fill Algorithm

- Figure 3-12 shows two methods for proceeding to neighboring pixels from the current test position.
- In Fig. 3-12(a), four neighboring points are tested. These are the pixel positions that are right, left, above, and below the current pixel. Areas filled by this method are called 4-connected.

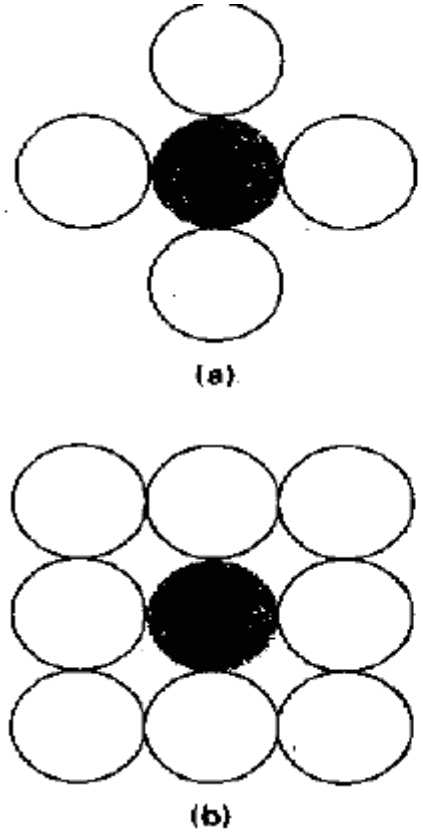


Figure 3-12

Fill methods applied to a 4-connected area (a) and to an 8-connected area (b). Open circles represent pixels to be tested from the current test position, shown as a solid color.

Boundary-Fill Algorithm

- The second method, shown in Fig. 3-12(b), is used to fill more complex figures. Here the set of neighboring positions to be tested includes the four diagonal pixels.
- Fill methods using this approach are called 8-connected. An 8-connected boundary-fill algorithm would correctly fill the interior of the area defined in Fig. 3-13,
- but a 4-connected boundary-fill algorithm produces the partial fill shown.

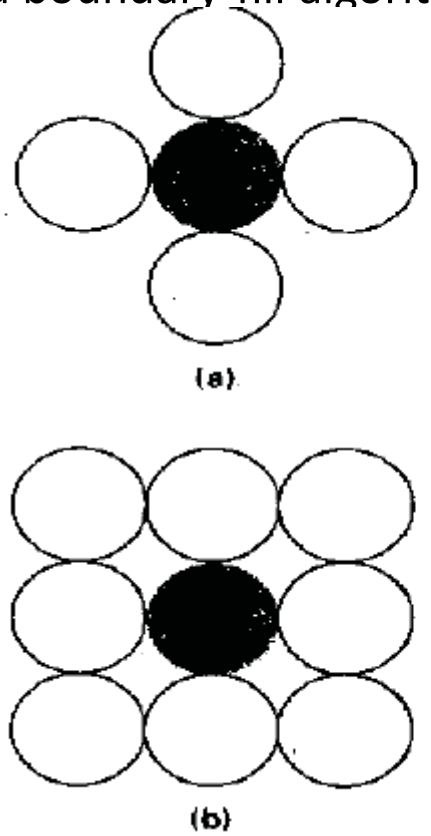


Figure 3-12

Fill methods applied to a 4-connected area (a) and to an 8-connected area (b). Open circles represent pixels to be tested from the current test position, shown as a solid color.

Boundary-Fill Algorithm

- The procedure illustrates a recursive method for filling a **4-connected area with** an intensity specified in parameter fill up to a boundary color specified with parameter boundary.
- We can extend this procedure to fill an 8-connected region by including four additional statements to test diagonal positions, such is (*x+1, y+1).*

```
void boundaryFill4 (int x, int y, int fill, int boundary)
{
    int current;
    current = getpixel (x, y);
    if ((current != boundary) && (current != fill))
    {
        setcolor (fill);
        setpixel (x, y);
        boundaryFill4 (x+1, y, fill, boundary);
        boundaryFill4 (x−1, y, fill, boundary);
        boundaryFill4 (x, y+l, fill, boundary);
        boundaryFill4 (x, y−1, fill, boundary);
    }
}
```

Boundary-Fill Algorithm

- Recursive boundary-fill algorithms may not fill regions correctly if some interior pixels are already displayed in the fill color.
- This occurs because the algorithm checks next pixels both for boundary color and for fill color.
- Encountering a pixel with the fill color can cause a recursive branch to terminate, leaving other interior pixels unfilled.
- To avoid this, we can first change the color of any interior pixels that are initially set to the fill color before applying the boundary-fill procedure.

Flood-Fill Algorithm

- Begin with a seed(starting pixel) inside the region.
- It checks to see if the pixel has the region's original color.
- If the answer is yes, it fills the pixel with a new color and uses each of the pixel's neighbors as anew seed in a recursive call.
- If answer is no , it returns to the caller.

Flood-Fill Algorithm

- We start from a specified interior point (*x, y) and reassign all pixel values that are currently* set to a given interior color with the desired fill color.
- If the area we want to paint has more than one interior color, we can first reassign pixel values so that all interior points have the same color.
- Using either a 4-connected or 8-connected approach, we then step through pixel positions until all interior points have been repainted.

Flood-Fill Algorithm

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

```
Void floodFill4 (int x, int y, int fillColor, int oldColor)
{
    if (getpixel (x, y) == oldcolor)
    {
        setcolor (fillcolor);
        setpixel (x, y);
        floodFill4 (x+l, y, fillColor, oldColor);
        floodFill4 (x-1, y, fillColor, oldColor);
        floodFill4 (x, y+l, fillColor, oldColor);
        floodFill4 (x, y-1, fillColor, oldColor);
    }
}
```