Region Fill Algorithms

– Seed Fill Approaches
  • Boundary Fill
  • Flood Fill

  Work at the pixel level. Suitable for interactive painting applications

– Scanline Fill Approaches

  Work at the polygon level

  Better performance
– Connectedness

- **4–connected region:**
  
  From a given pixel, the region that you can get to by a series of 4 way moves (north, south, east, west)

- **8–connected region:**
  
  From a given pixel, the region that you can get to by a series of 8 way moves (north, south, east, west, NE, NW, SE, SW)
Boundary Fill Algorithm

- Start at a point inside a region
- Paint the interior outward toward the boundary
- The boundary is specified in a single color
- Fill the 4–connected or 8–connected region

```java
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+1, fill, boundary);
        boundaryFill4(x, y−1, fill, boundary);
    }
}
```
4–connected fill is faster, but can have problems:
Flood Fill Algorithm

- Used when an area defined with multiple color boundaries

- Start at a point inside a region

- Replace a specified interior color (old color) with fill color

- Fill the 4-connected or 8-connected region until all interior points being replaced

```cpp
void floodFill4(int x, int y, int fill, int oldColor) {
    if (getPixel(x, y) == oldColor) {
        setColor(fill);
        setPixel(x, y);
        floodFill4(x+1, y, fill, oldColor);
        floodFill4(x-1, y, fill, oldColor);
        floodFill4(x, y+1, fill, oldColor);
        floodFill4(x, y-1, fill, oldColor);
    }
}
```