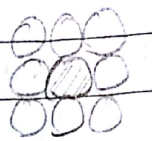
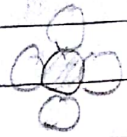
Filling algorithms
- Scan line
- Flood fill

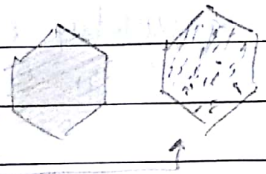

8-neighbourhood


4-neighbourhood

x, y
x+1, y x+1, y+1
x-1, y x+1, y-1
x, y+1 x-1, y+1
x, y-1 x-1, y-1

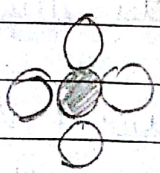
Lecture 15
Mon
28/02/18

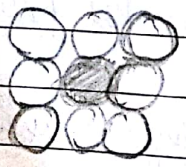
Filled area primitives
- Solid fill
- Pattern fill
- Polygon of n-vertices, ordered list of edges



Filling algorithms
- Scan-line fill
- Flood fill
- Boundary fill

Boundary fill algorithms
- To start from an interior point and paint the interior outward towards the boundary
- Two types:

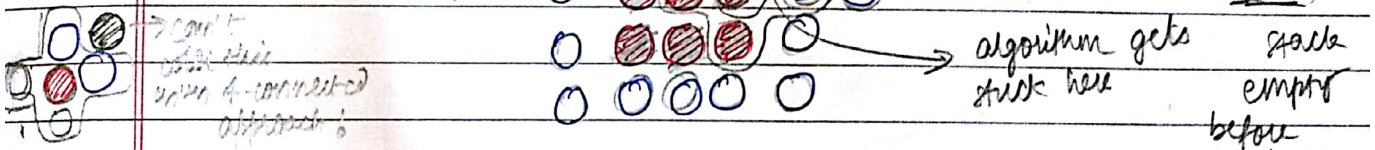

4-neighbourhood


8-neighbourhood

```
void boundaryfill (int x, int y, int fillColor, int borderColor)
{
    getPixel(x, y, color);
    if ((color != borderColor) && (color != fillColor)) {
        setPixel(x, y);
        boundaryfill (x+1, y, fillColor, borderColor);
        boundaryfill (x-1, y, fillColor, borderColor);
        boundaryfill (x, y+1, fillColor, borderColor);
        boundaryfill (x, y-1, fillColor, borderColor);
    }
}
```

recursively filling the neighbours

Drawback of 4-neighbourhood method :(



- 8 connected method can solve this problem

Flood filled algorithm

- To fill an area or recolor it whose boundary is not defined by a single color.
- Paint by replacing color instead of checking for boundary color.
- If more than one interior color, first reassign to a single color.
- 4 connected or 8-connected approach.

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