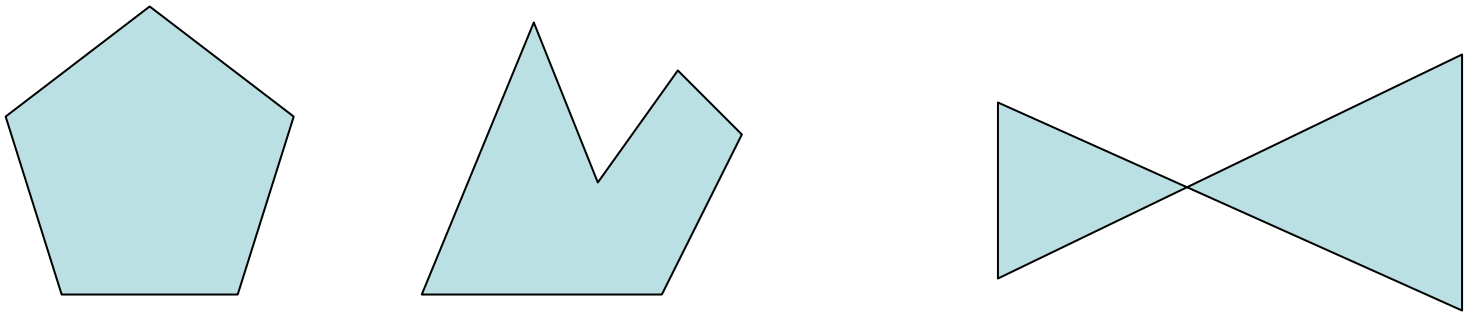
Fill Area Algorithms

Jan. 2008

Polygon Fill Algorithm

- ***Different types of Polygons***
  - Simple Convex
  - Simple Concave
  - Non-simple : self-intersecting
  - With holes



Convex Concave Self-intersecting

Polygon Fill Algorithm

- A scan-line fill algorithm of a region is performed as follows:
  1. Determining the intersection positions of the boundaries of the fill region with the screen scan lines.
  2. Then the fill colors are applied to each section of a scan line that lies within the interior of the fill region.
- The simplest area to fill is a polygon, because each scan-line intersection point with a polygon boundary is obtained by solving a pair of simultaneous linear equations, where the equation for the scan line is simply y = constant.

Example

- Consider the following polygon:

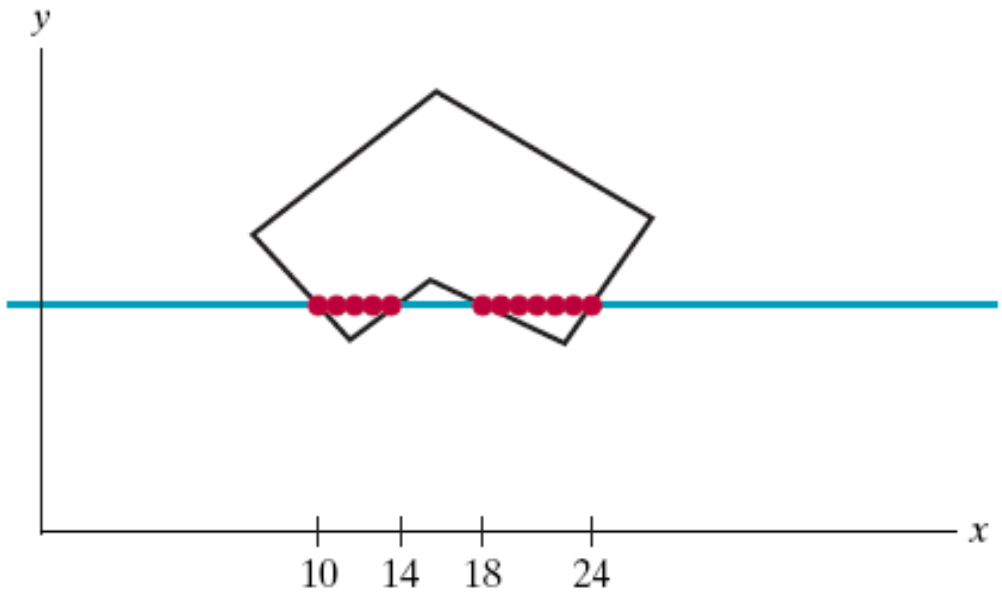


FIGURE 4–20 Interior pixels along a scan line passing through a polygon fill area.

Example

- For each scan line that crosses the polygon, the edge intersections are sorted from left to right, and then the pixel positions between, and including, each intersection pair are set to the specified fill color.

- In the previous Figure, the four pixel intersection positions with the polygon boundaries define two stretches of interior pixels.

Example

- The fill color is applied to the five pixels:
  - from $x = 10$ to $x = 14$

and

- To the seven pixels
  - from $x = 18$ to $x = 24$.

Polygon Fill Algorithm

- However, the scan-line fill algorithm for a polygon is not quite as simple
- Whenever a scan line passes through a vertex, it intersects two polygon edges at that point.
- In some cases, this can result in an odd number of boundary intersections for a scan line.

Polygon Fill Algorithm

- Consider the next Figure.
- It shows two scan lines that cross a polygon fill area and intersect a vertex.
- Scan line *y'* intersects an even number of edges, and the two pairs of intersection points along this scan line correctly identify the interior pixel spans.
- But scan line *y* intersects five polygon edges.

Polygon Fill Algorithm

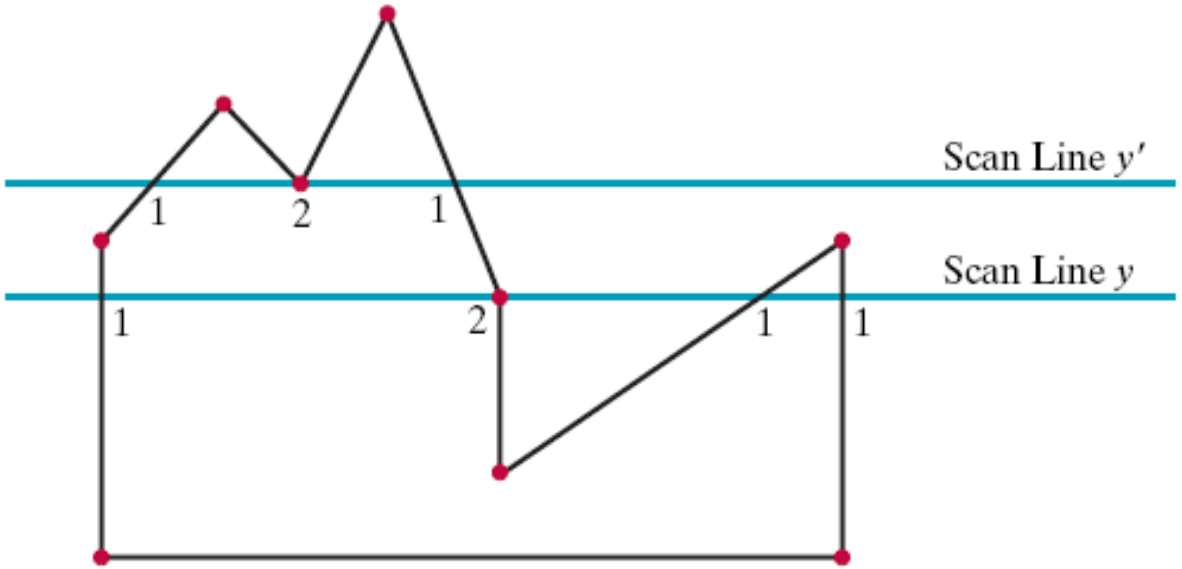


FIGURE 4–21 Intersection points along scan lines that intersect polygon vertices. Scan line y generates an odd number of intersections, but scan line y' generates an even number of intersections that can be paired to identify correctly the interior pixel spans.

Polygon Fill Algorithm

- To identify the interior pixels for scan line *y*, we must count the vertex intersection as only one point.

- Thus, as we process scan lines, we need to distinguish between these two cases.

Polygon Fill Algorithm

- We can detect the difference between the two cases by noting the position of the intersecting edges relative to the scan line.
- For scan line *y*, the two edges sharing an intersection vertex are on opposite sides of the scan line.
- But for scan line *y'*, the two intersecting edges are both above the scan line.

Polygon Fill Algorithm

- A vertex that has adjoining edges on opposite sides of an intersecting scan line should be counted as just one boundary intersection point.

- We can identify these vertices by tracing around the polygon boundary in either clockwise or counterclockwise order and observing the relative changes in vertex *y* coordinates as we move from one edge to the next.

Polygon Fill Algorithm

- If the three endpoint *y* values of two consecutive edges increase or decrease, we need to count the shared (middle) vertex as a single intersection point for the scan line passing through that vertex.

- Otherwise, the shared vertex represents a local extremum (minimum or maximum) on the polygon boundary, and the two edge intersections with the scan line passing through that vertex can be added to the intersection list.

Area Fill Algorithm

- An alternative approach for filling an area is to start at a point inside the area and “paint” the interior, point by point, out to the boundary.
- This is a particularly useful technique for filling areas with irregular borders, such as a design created with a paint program.
- The algorithm makes the following assumptions
  - one interior pixel is known, and
  - pixels in boundary are known.

Area Fill Algorithm

- If the boundary of some region is specified in a single color, we can fill the interior of this region, pixel by pixel, until the boundary color is encountered.
- This method, called the **boundary-fill algorithm,** is employed in interactive painting packages, where interior points are easily selected.

Example

- One can sketch a figure outline, and pick an interior point.
- The figure interior is then painted in the fill color as shown in these Figures

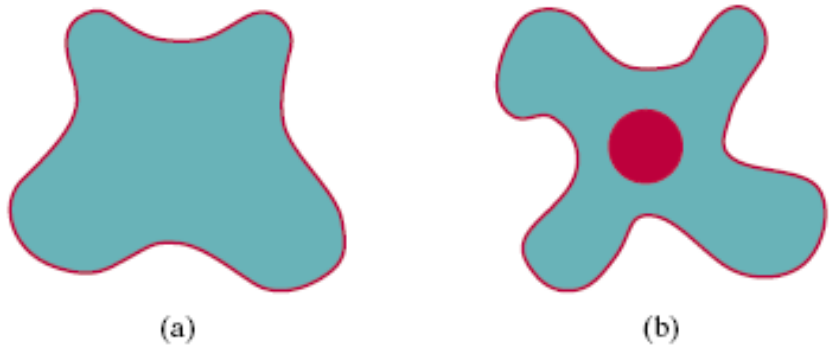


FIGURE 4–26 Example color boundaries for a boundary-fill procedure.

Area Fill Algorithm

- Basically, a boundary-fill algorithm starts from an interior point (*x*, *y*) and sets the neighboring points to the desired color.

- This procedure continues until all pixels are processed up to the designated boundary for the area.

Area Fill Algorithm

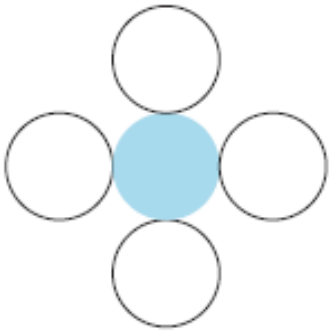
- There are two methods for processing neighboring pixels from a current point.

1. Four neighboring points.
   - These are the pixel positions that are right, left, above, and below the current pixel.
   - Areas filled by this method are called **4-connected.**

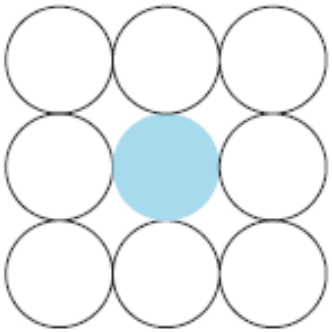
Area Fill Algorithm

2. Eight neighboring points.
   - This method is used to fill more complex figures.
   - Here the set of neighboring points to be set includes the four diagonal pixels, in addition to the four points in the first method.
   - Fill methods using this approach are called **8-connected.**

Area Fill Algorithm



(a)



(b)

FIGURE 4–27 Fill methods applied to a 4-connected area (a) and to an 8-connected area (b). Hollow circles represent pixels to be tested from the current test position, shown as a solid color.

Area Fill Algorithm

- Consider the Figure in the next slide.
- An 8-connected boundary-fill algorithm would correctly fill the interior of the area defined in the Figure.

- But a 4-connected boundary-fill algorithm would only fill part of that region.

Area Fill Algorithm

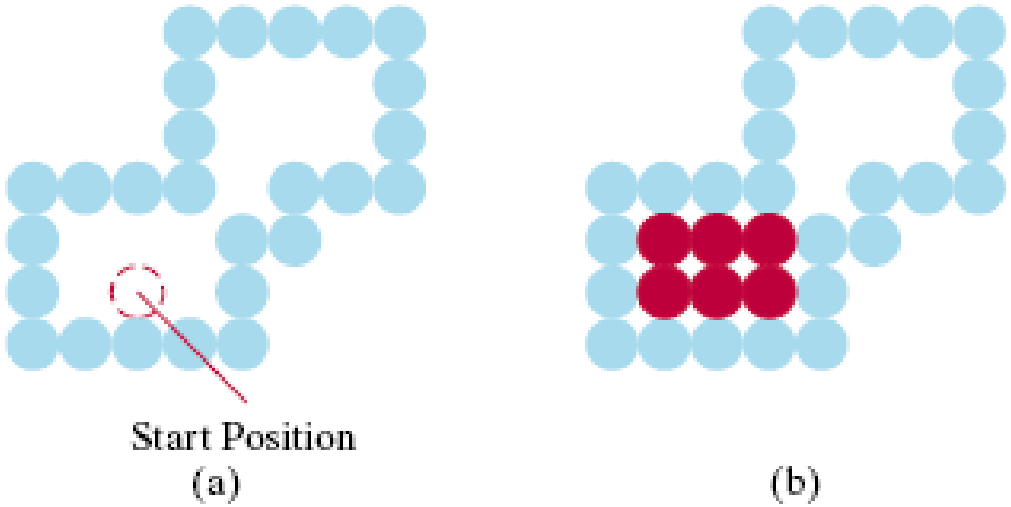


FIGURE 4-28 The area defined within the color boundary (a) is only partially filled in (b) using a 4-connected boundary-fill algorithm.

Area Fill Algorithm

- The following procedure illustrates a recursive method for painting a 4-connected area with a solid color, specified in parameter **fillColor**, up to a boundary color specified with parameter **borderColor**.
- We can extend this procedure to fill an 8-connected region by including four additional statements to test the diagonal positions ($x \pm 1$, $y \pm 1$).

Area Fill Algorithm

```
void boundaryFill4 (int x, int y, int fillColor, int borderColor)
{
   int interiorColor;

   /* Set current color to fillColor, then perform following oprations. */
   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)
   }
}
```

Area Fill Algorithm

- Some times we want to fill in (or recolor) an area that is not defined within a single color boundary.
- Consider the following Figure.

FIGURE 4–30 An area defined within multiple color boundaries.

Area Fill Algorithm

- We can paint such areas by replacing a specified interior color instead of searching for a particular boundary color.

- This fill procedure is called a **flood-fill algorithm.**

Area 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.

Area Fill Algorithm

- Using either a 4-connected or 8-connected approach, we then step through pixel positions until all interior points have been repainted.

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

Area 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)
   }
}
```

Problems with Fill Algorithm (1)

- 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.

Problems with Fill Algorithm

- 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.

- Encountering a pixel with the fill color can cause a recursive branch to terminate, leaving other interior pixels unfilled.

Problems with Fill Algorithm (2)

- This procedure requires considerable stacking of neighboring points, more efficient methods are generally employed.
- These methods fill horizontal pixel spans across scan lines, instead of proceeding to 4-connected or 8-connected neighboring points.

Problems with Fill Algorithm (2)

- Then we need only stack a beginning position for each horizontal pixel span, instead of stacking all unprocessed neighboring positions around the current position.
- Starting from the initial interior point with this method, we first fill in the contiguous span of pixels on this starting scan line.

Problems with Fill Algorithm (2)

- Then we locate and stack starting positions for spans on the adjacent scan lines, where spans are defined as the contiguous horizontal string of positions bounded by pixels displayed in the border color.
- At each subsequent step, we retrieve the next start position from the top of the stack and repeat the process.

Area Fill Algorithm

The algorithm can be summarized as follows:

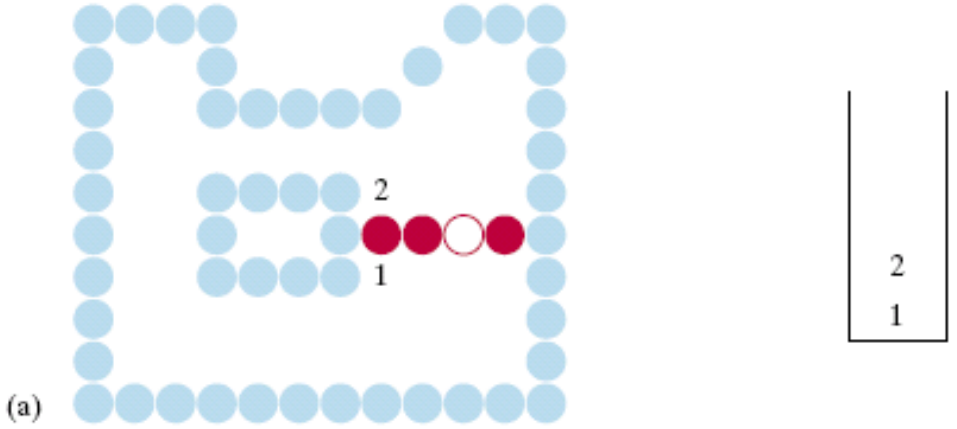
1. define seed point,
2. fill scan line containing seed point,
3. for scan lines above and below, define new seed points as:
    i) first point inside left boundary,
    ii) subsequent points within boundary whose left neighbor is outside,
4. d) repeat algorithm with the new set of seed points.

Example

- In this example, we first process scan lines successively from the start line to the top boundary.
- After all upper scan lines are processed, we fill in the pixel spans on the remaining scan lines in order down to the bottom boundary.
- The leftmost pixel position for each horizontal span is located and stacked, in left to right order across successive scan lines.

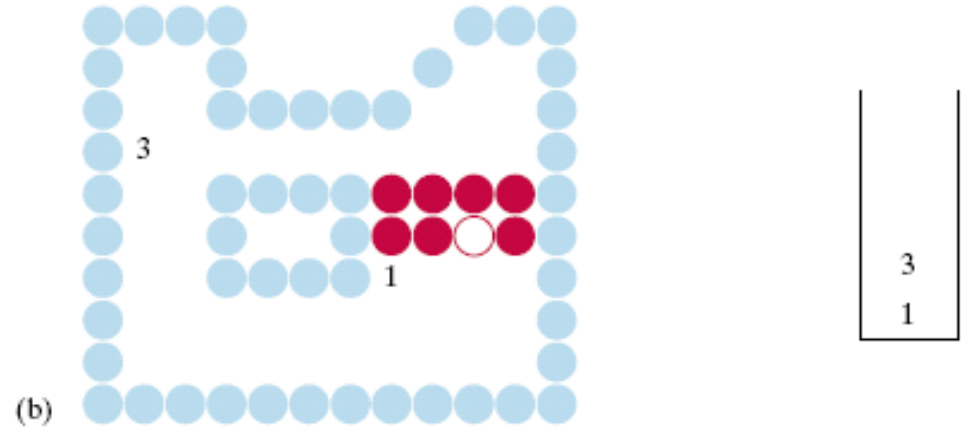
Example

- In (a) of this figure, the initial span has been filled, and starting positions 1 and 2 for spans on the next scan lines (below and above) are stacked.



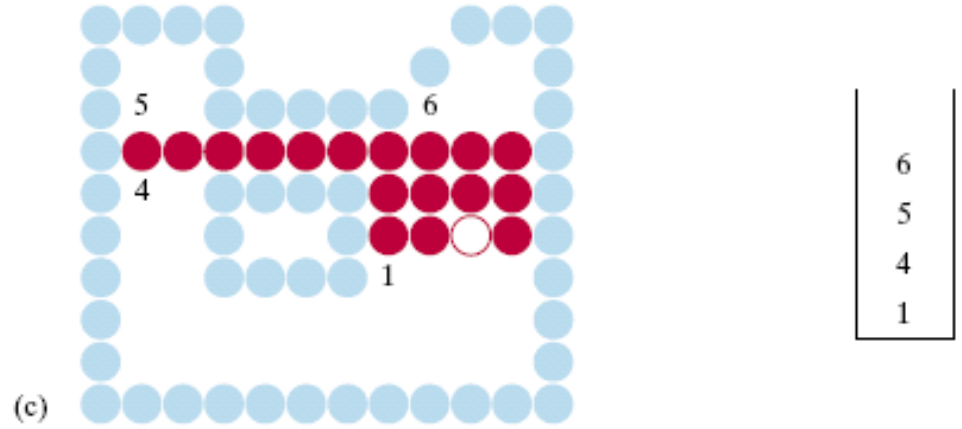
Example

- In Fig.(b), position 2 has been unstacked and processed to produce the filled span shown, and the starting pixel (position 3) for the single span on the next scan line has been stacked.



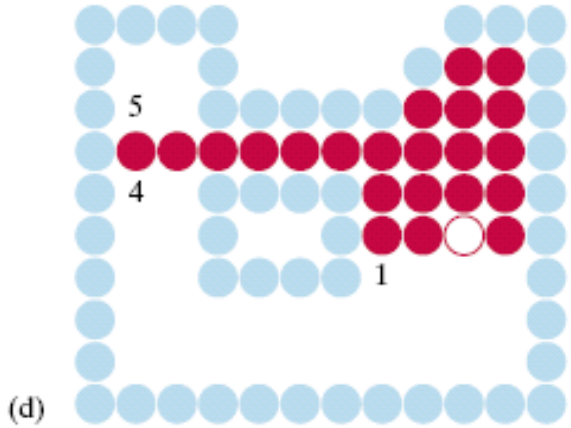
Example

- After position 3 is processed, the filled spans and stacked positions are as shown in Fig. (c).



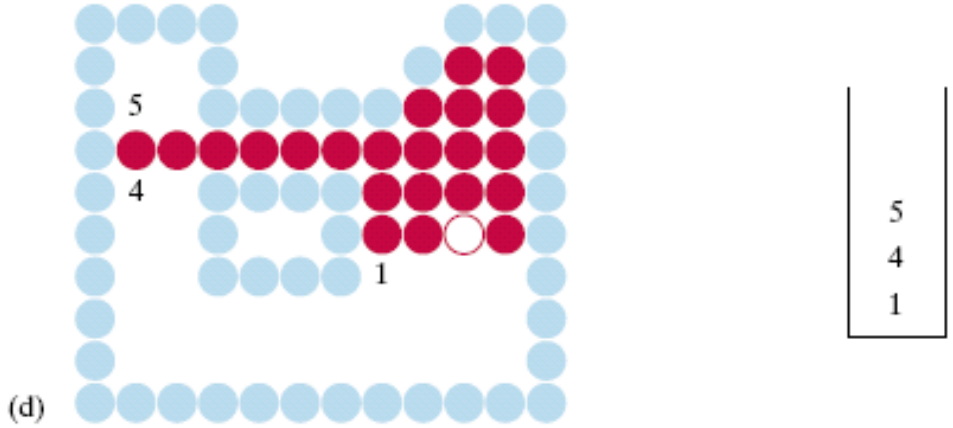
Example

- And Fig.(d) shows the filled pixels after processing all spans in the upper right of the specified area.



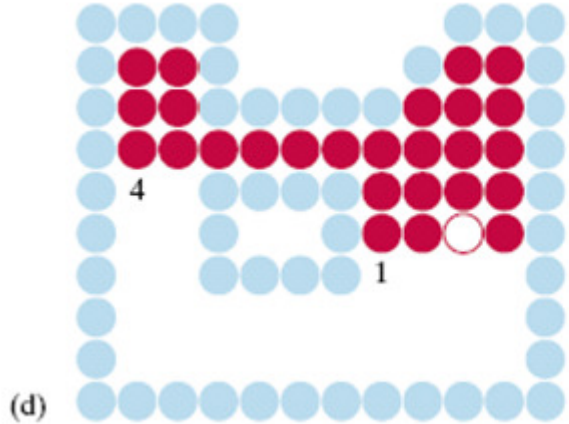
Example

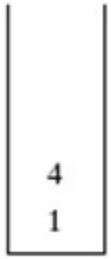
- Position 5 is next processed, and spans are filled in the upper left of the region; then position 4 is picked up to continue the processing for the lower scan lines.



Example

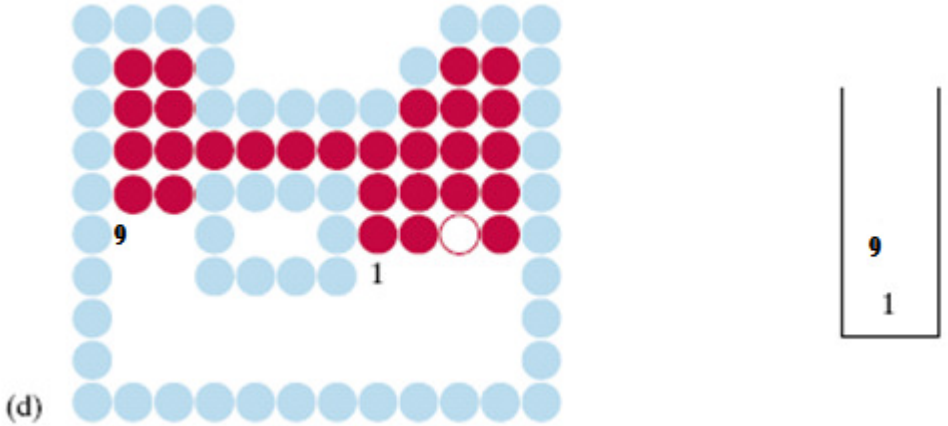
- Finish up the upper scan lines.





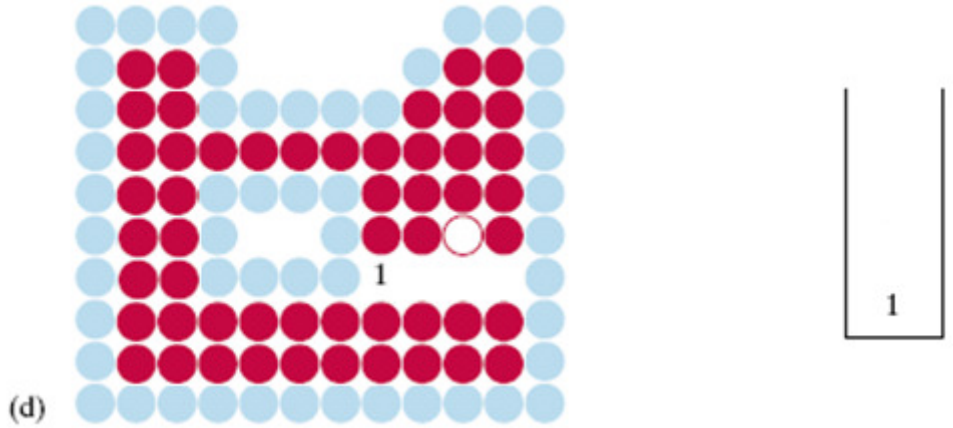
Example

- Start the bottom scan lines.



Example

- Finish up the bottom scan lines.



Example

- Finish up the bottom scan lines.