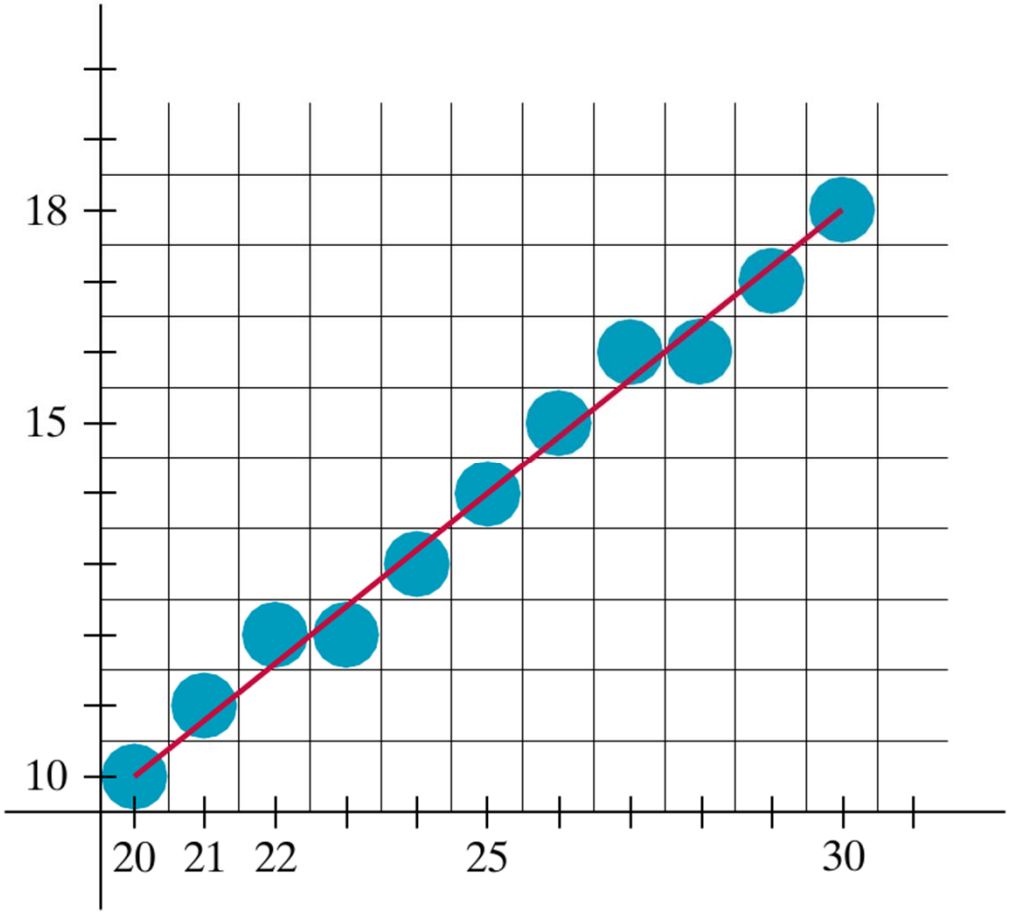
Chapter 6
Implementation Algorithms

Myung-Soo Kim
Seoul National University
http://cse.snu.ac.kr/mskim
http://3map.snu.ac.kr

Line Drawing



Line Drawing Algorithms

$$y = m \cdot x + b \qquad (x_0, y_0) \text{ and } (x_{\text{end}}, y_{\text{end}})$$

$$m = \frac{y_{\text{end}} - y_0}{x_{\text{end}} - x_0} \qquad b = y_0 - m \cdot x_0$$

$$\delta y = m \cdot \delta x \qquad y_{k+1} = y_k + m$$

$$\delta x = \frac{\delta y}{m} \qquad x_{k+1} = x_k + \frac{1}{m}$$

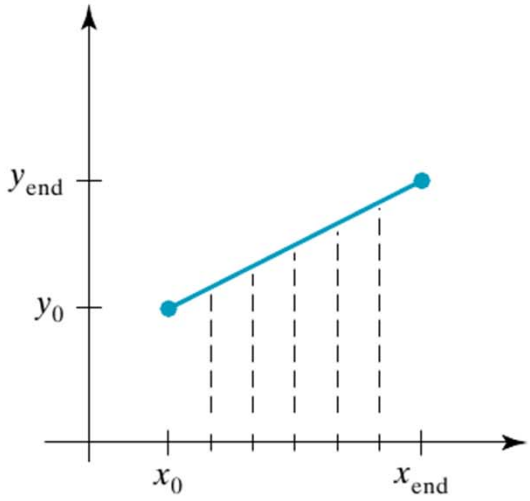


FIGURE 3-7 Straight-line segment with five sampling positions along the x axis between x_0 and x_{end}.

DDA Algorithm

```
void lineDDA (int x0, int y0, int xEnd, int yEnd)
{
    int dx = xEnd - x0,  dy = yEnd - y0,  steps,  k;
    float xIncrement, yIncrement, x = x0, y = y0;

    if (fabs (dx) > fabs (dy))
       steps = fabs (dx);
    else
       steps = fabs (dy);
    xIncrement = float (dx) / float (steps);
    yIncrement = float (dy) / float (steps);

    setPixel (round (x), round (y));
    for (k = 0; k < steps; k++) {
       x += xIncrement;
       y += yIncrement;
       setPixel (round (x), round (y));
    }
}
```

Bresenham Algorithm

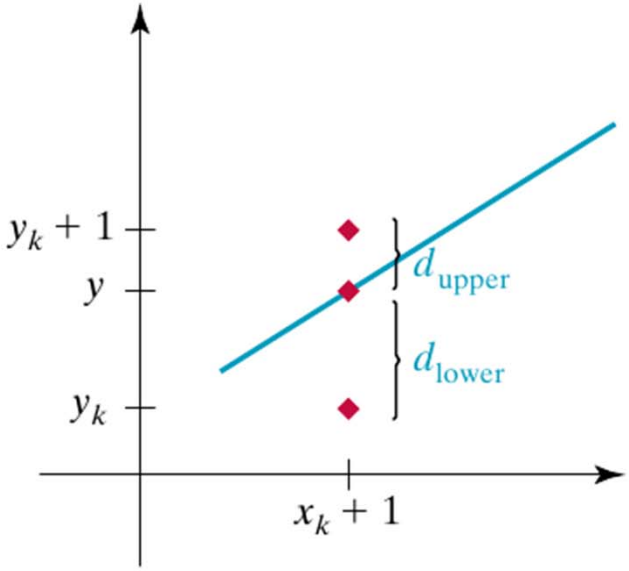
$$y = m(x_k + 1) + b$$

$$d_{\text{lower}} = y - y_k$$

$$= m(x_k + 1) + b - y_k$$

$$d_{\text{upper}} = (y_k + 1) - y$$

$$= y_k + 1 - m(x_k + 1) - b$$

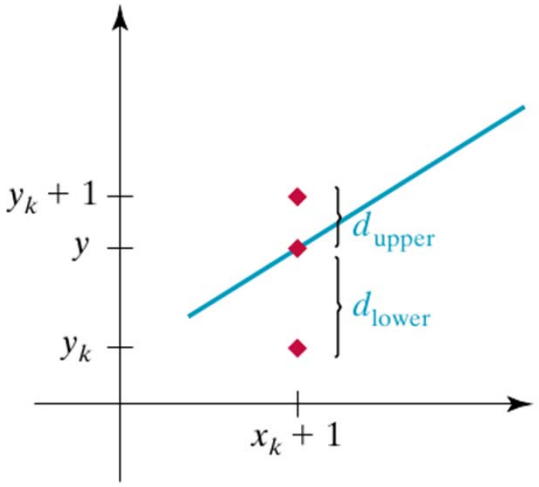


Bresenham Algorithm

$$y = m(x_k + 1) + b$$

$$\begin{aligned} d_{\text{lower}} &= y - y_k \\ &= m(x_k + 1) + b - y_k \end{aligned}$$

$$\begin{aligned} d_{\text{upper}} &= (y_k + 1) - y \\ &= y_k + 1 - m(x_k + 1) - b \end{aligned}$$



$$d_{\text{lower}} - d_{\text{upper}} = 2m(x_k + 1) - 2y_k + 2b - 1$$

$$p_k = \Delta x(d_{\text{lower}} - d_{\text{upper}})$$

$$= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$$

Bresenham Algorithm

$$d_{\text{lower}} - d_{\text{upper}} = 2m(x_k + 1) - 2y_k + 2b - 1$$

$$p_k = \Delta x(d_{\text{lower}} - d_{\text{upper}})$$

$$= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$$

$$p_{k+1} = 2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + c$$

$$p_{k+1} - p_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$$

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

$$p_0 = 2\Delta y - \Delta x$$

Bresenham's Line-Drawing Algorithm for $|m| < 1.0$

1. Input the two line endpoints and store the left endpoint in (x_0, y_0).
2. Set the color for frame-buffer position (x_0, y_0); i.e., plot the first point.
3. Calculate the constants Δx, Δy, $2\Delta y$, and $2\Delta y - 2\Delta x$, and obtain the starting value for the decision parameter as

$$p_0 = 2\Delta y - \Delta x$$

4. At each x_k along the line, starting at $k = 0$, perform the following test. If $p_k < 0$, the next point to plot is $(x_k + 1, y_k)$ and

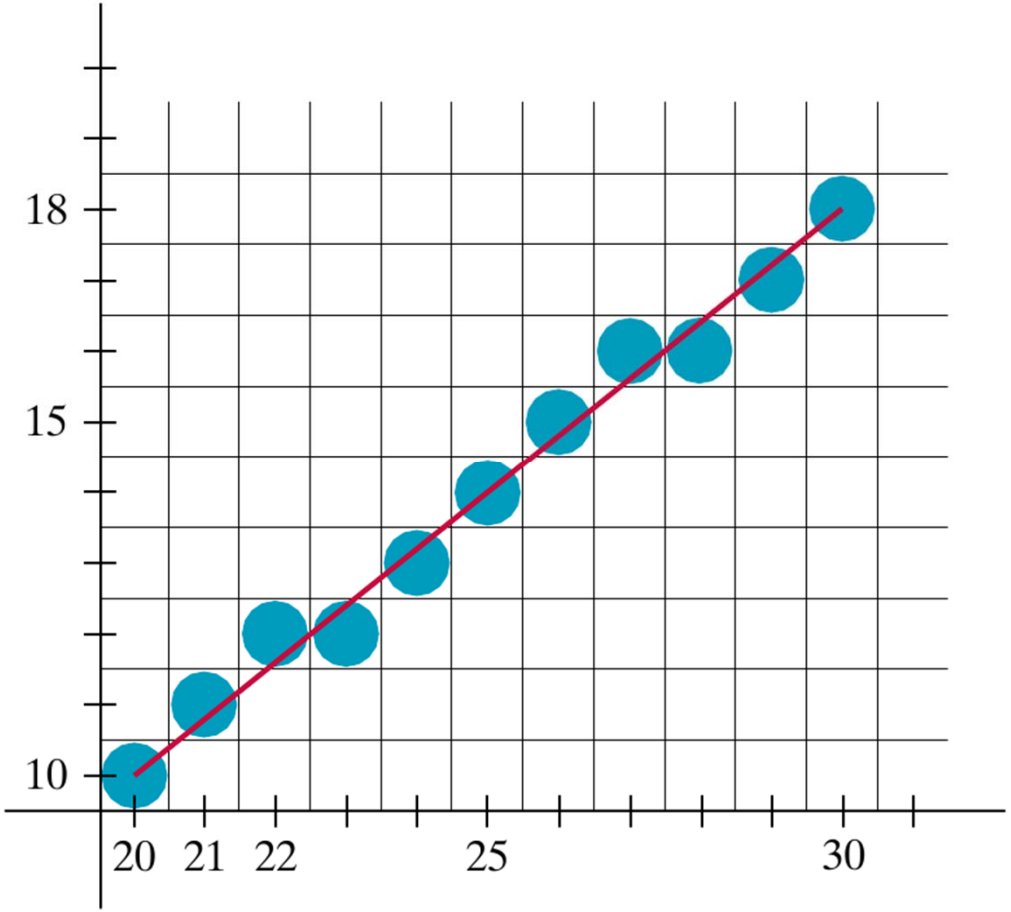
$$p_{k+1} = p_k + 2\Delta y$$

   Otherwise, the next point to plot is $(x_k + 1, y_k + 1)$ and

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

5. Perform step 4 $\Delta x - 1$ times.

Bresenham Algorithm



Pixel Addressing

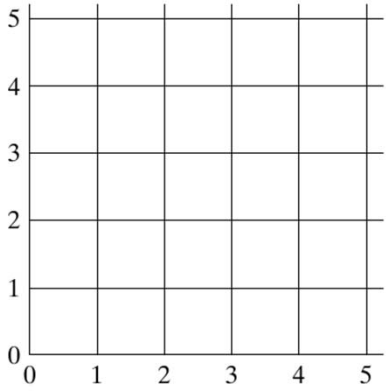


FIGURE 3-33 Lower-left section of a screen area with coordinate positions referenced by grid intersection lines.

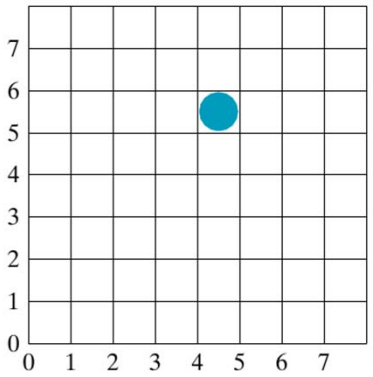


FIGURE 3-34 Illuminated pixel at raster position (4, 5).

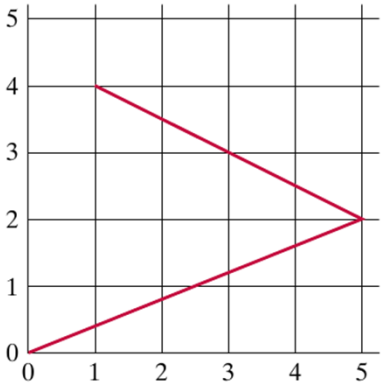


FIGURE 3-35 Line path for two connected line segments between screen grid-coordinate positions.

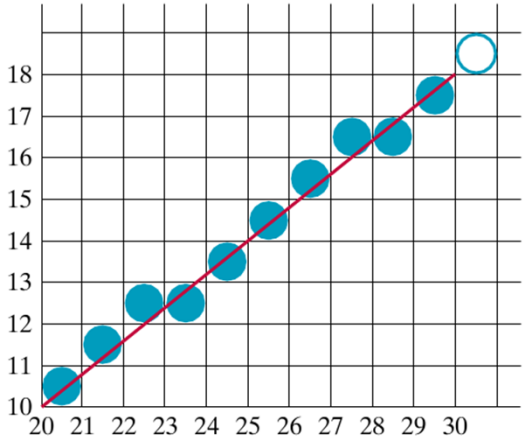
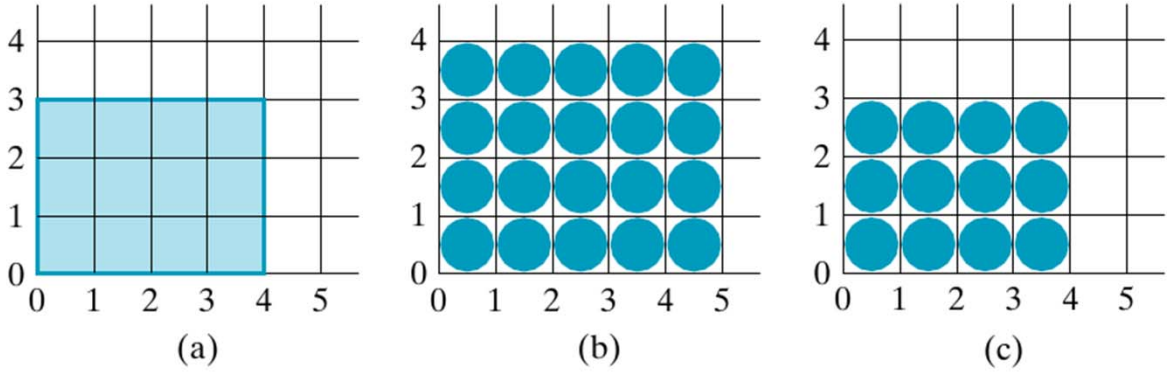


FIGURE 3-36 Line path and corresponding pixel display for grid endpoint coordinates (20, 10) and (30, 18).

FIGURE 3-37
Conversion of rectangle (a) with vertices at screen coordinates (0, 0), (4, 0), (4, 3), and (0, 3) into display (b) that includes the right and top boundaries and into display (c) that maintains geometric magnitudes.



Pixel Addressing

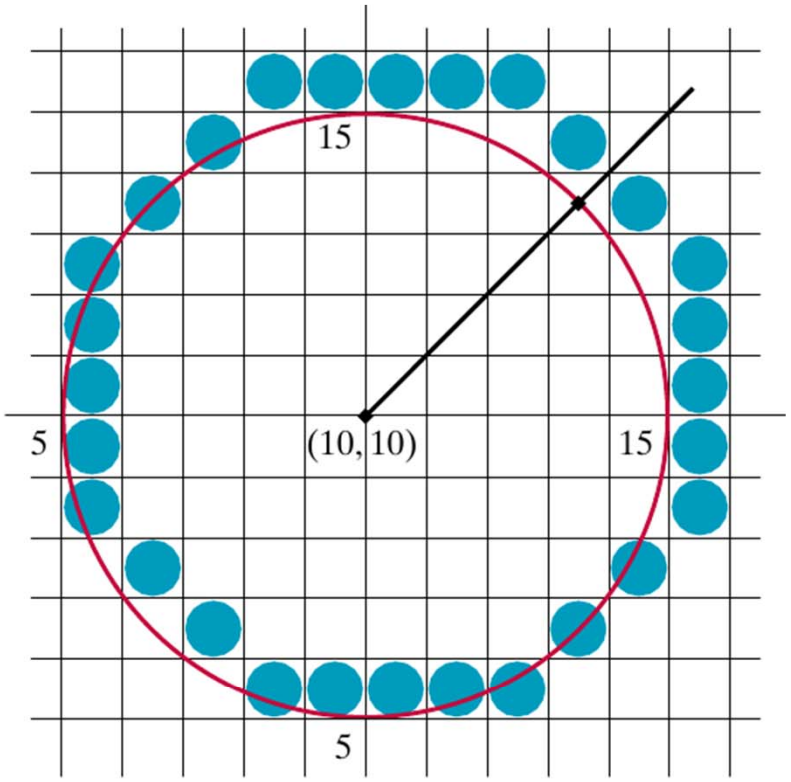


FIGURE 3-38
A midpoint-algorithm plot of the circle equation $(x - 10)^2 + (y - 10)^2 = 5^2$ using pixel-center coordinates.

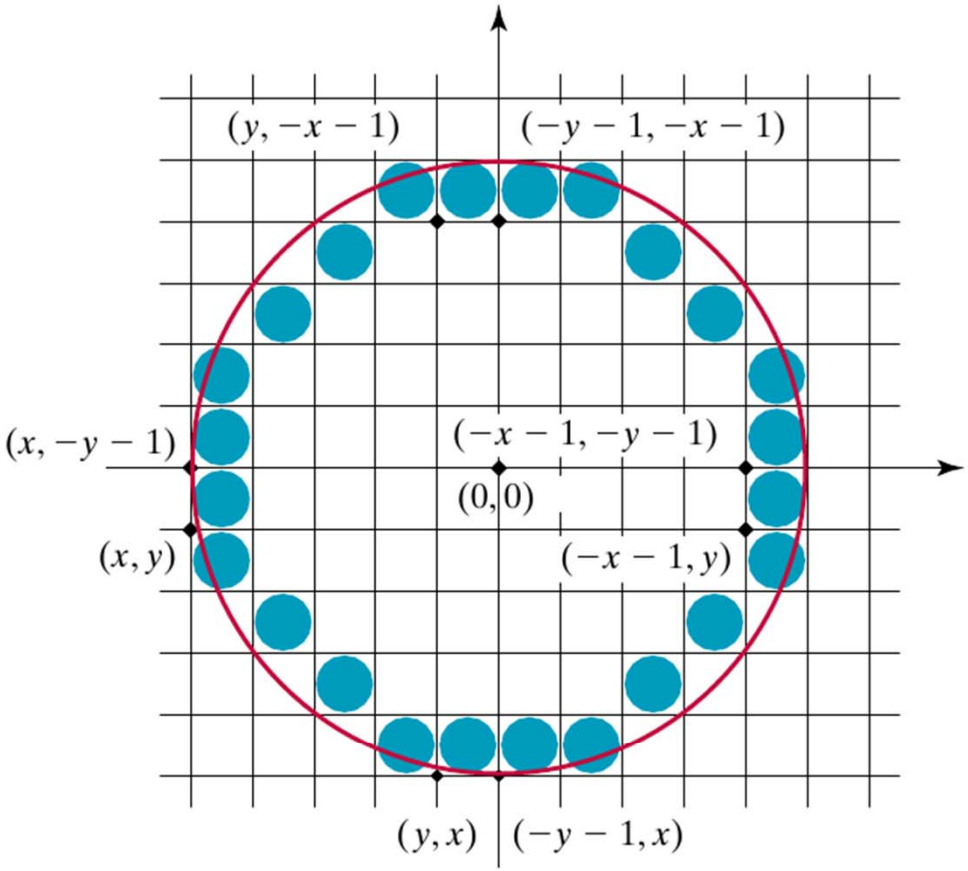


FIGURE 3-39
Modification of the circle plot in Fig. 3-38 to maintain the specified circle diameter of 10.

Scan-Line Polygon Fill

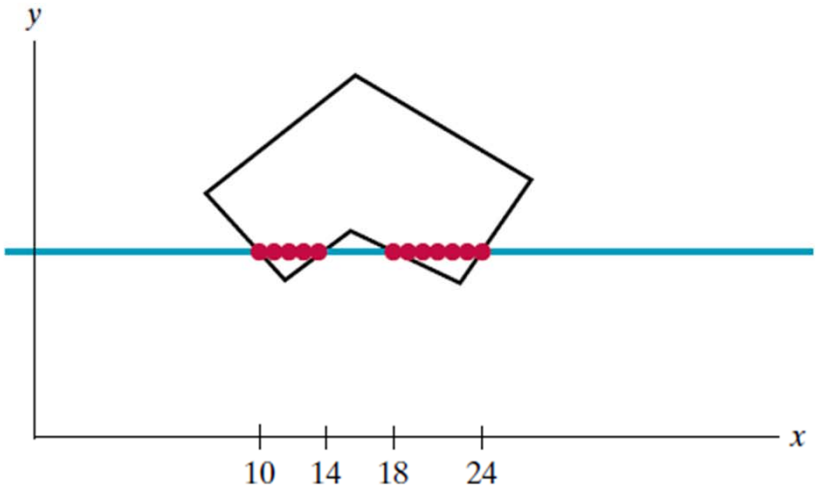


FIGURE 4-20 Interior pixels along a scan line passing through a polygon fill area.

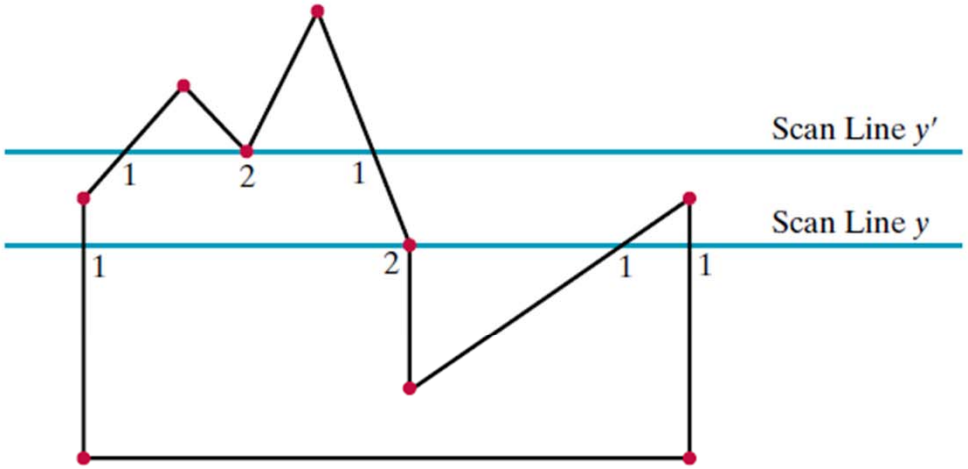


FIGURE 4-21 Intersection points along scan lines that intersect polygon vertices. Scan line y generates an odd number of intersections, but scan line y' generates an even number of intersections that can be paired to identify correctly the interior pixel spans.

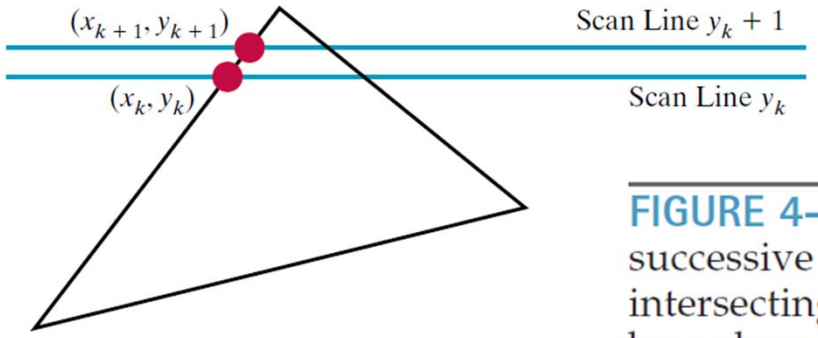


FIGURE 4-23 Two successive scan lines intersecting a polygon boundary.

Scan-Line Polygon Fill

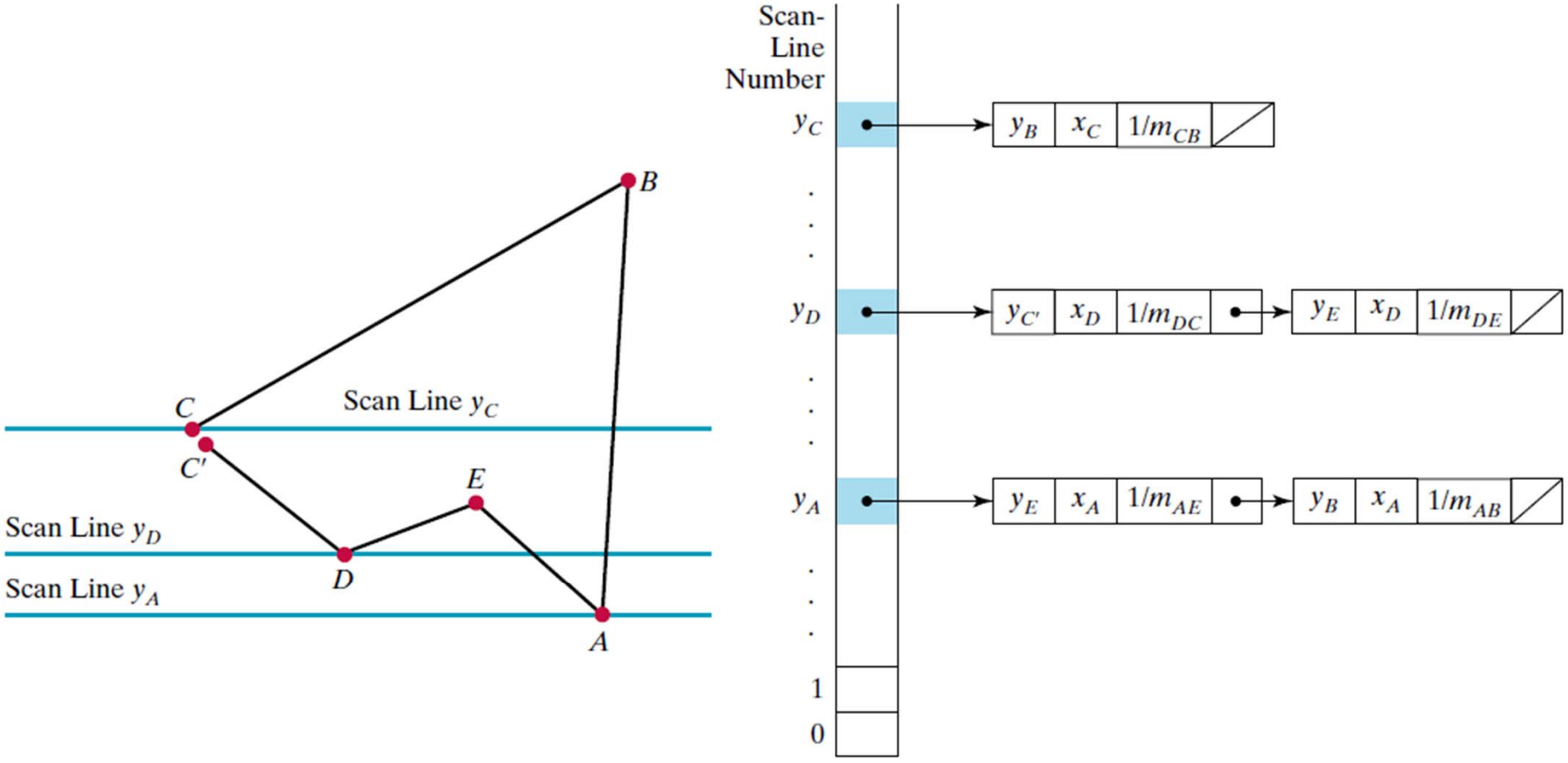


FIGURE 4-24 A polygon and its sorted edge table, with edge $\overline{DC}$ shortened by one unit in the y direction.

Boundary-Fill Algorithm

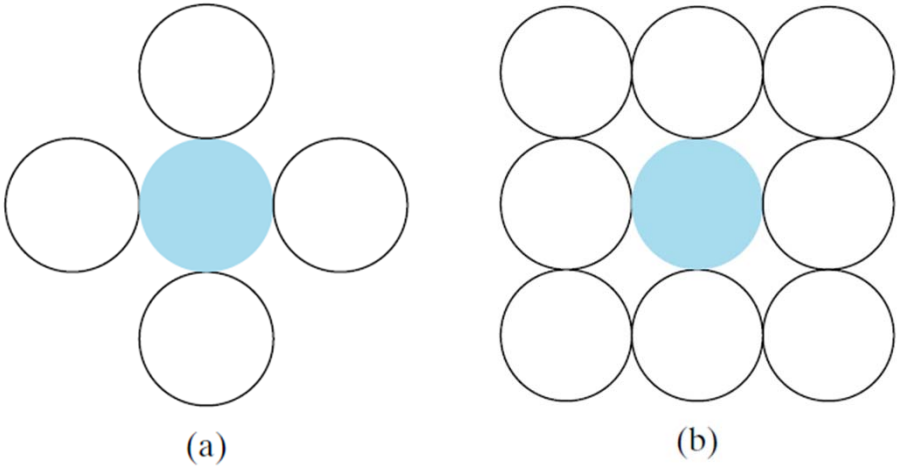


FIGURE 4-27 Fill methods applied to a 4-connected area (a) and to an 8-connected area (b). Hollow circles represent pixels to be tested from the current test position, shown as a solid color.

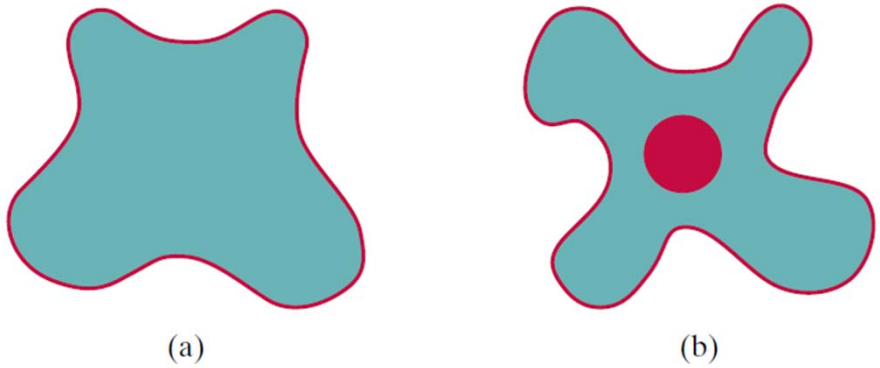


FIGURE 4-26 Example color boundaries for a boundary-fill procedure.

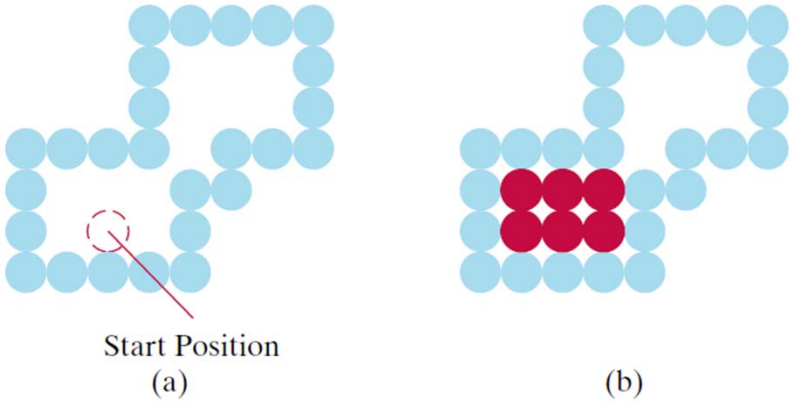


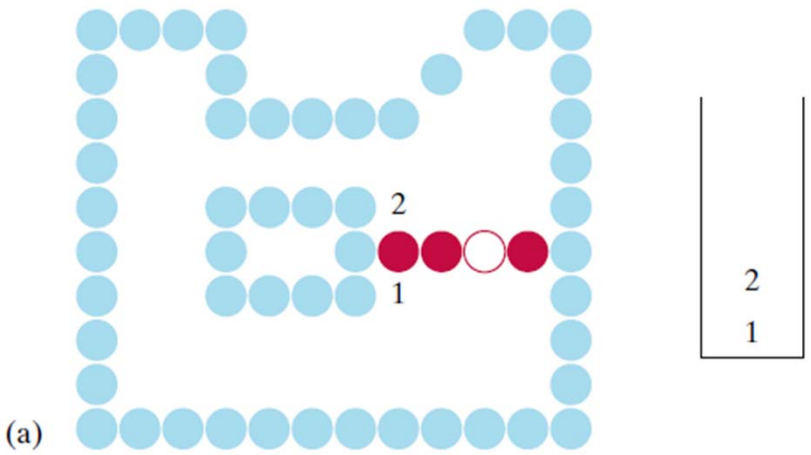
FIGURE 4-28 The area defined within the color boundary (a) is only partially filled in (b) using a 4-connected boundary-fill algorithm.

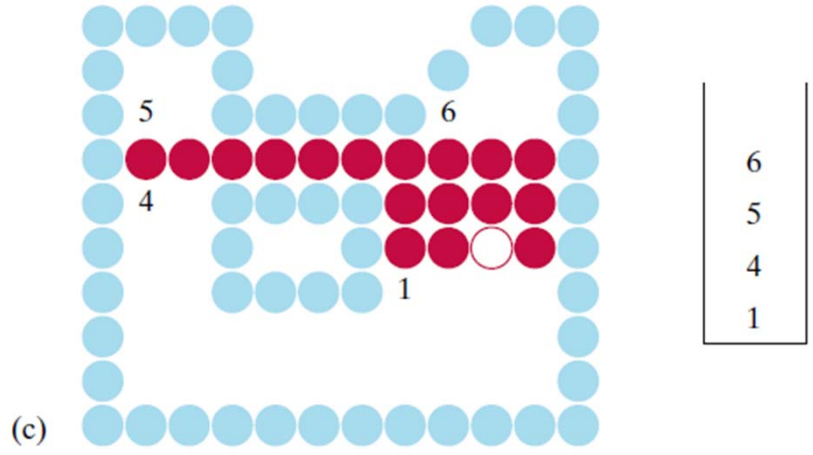
Boundary-Fill Algorithm

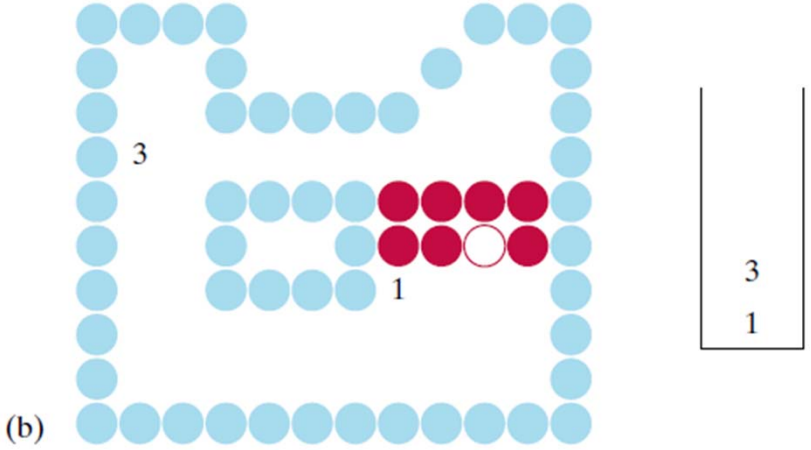
```
void boundaryFill4 (int x, int y, int fillColor, int borderColor)
{
   int interiorColor;
   getPixel (x, y, interiorColor);
   if ((interiorColor != borderColor) && (interiorColor != fillColor)) {
      setPixel (x, y);    // Set color of pixel to fillColor.
      boundaryFill4 (x + 1, y , fillColor, borderColor);
      boundaryFill4 (x - 1, y , fillColor, borderColor);
      boundaryFill4 (x , y + 1, fillColor, borderColor);
      boundaryFill4 (x , y - 1, fillColor, borderColor)
   }
}
```

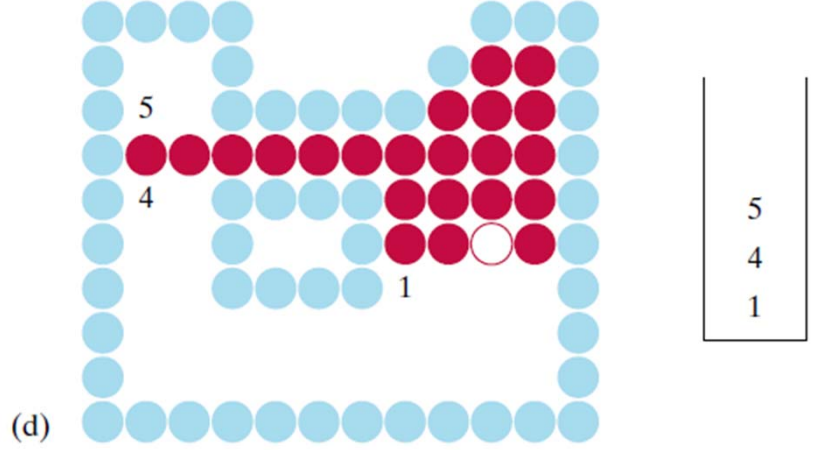
- This procedure requires considerable stacking of neighboring points

More Efficient Algorithm









Flood-Fill Algorithm

```
void floodFill4 (int x, int y, int fillColor, int interiorColor)
{
   int color;

   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)
   }
}
```

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

Antialiasing

- Information loss due to under-sampling

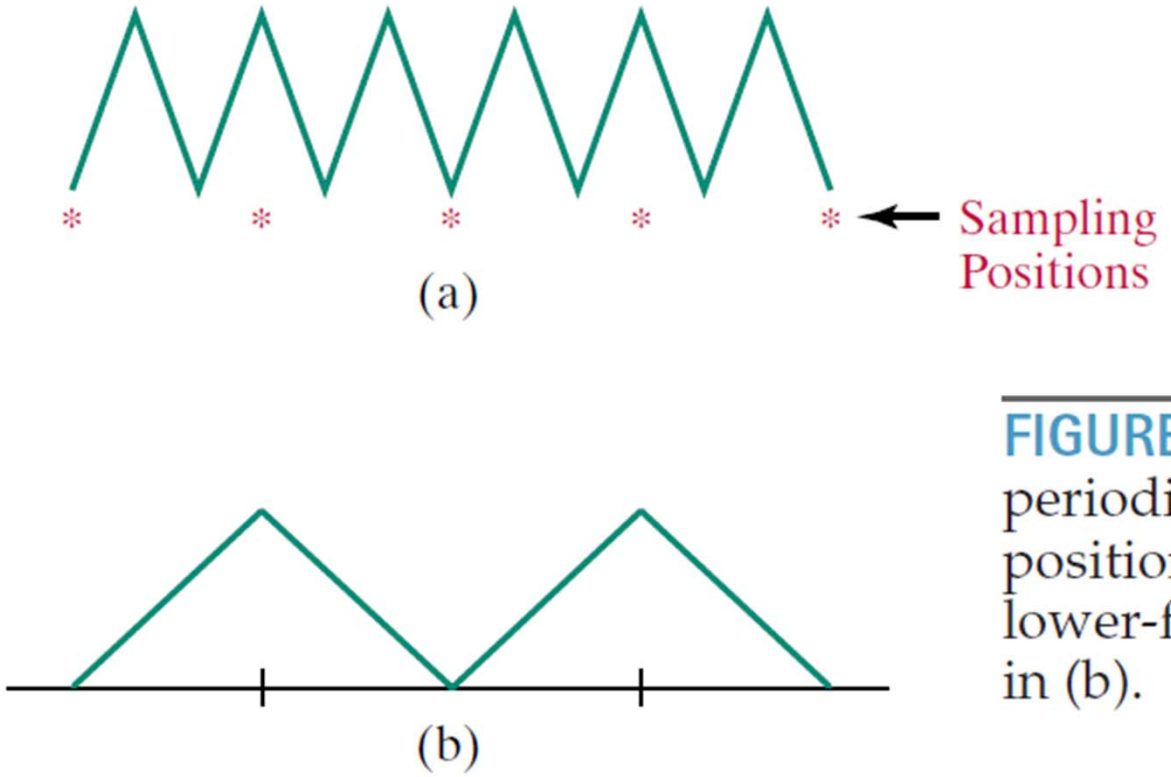


FIGURE 4-46 Sampling the periodic shape in (a) at the indicated positions produces the aliased lower-frequency representation in (b).

Antialiasing

- Super-sampling Straight-Line Segments

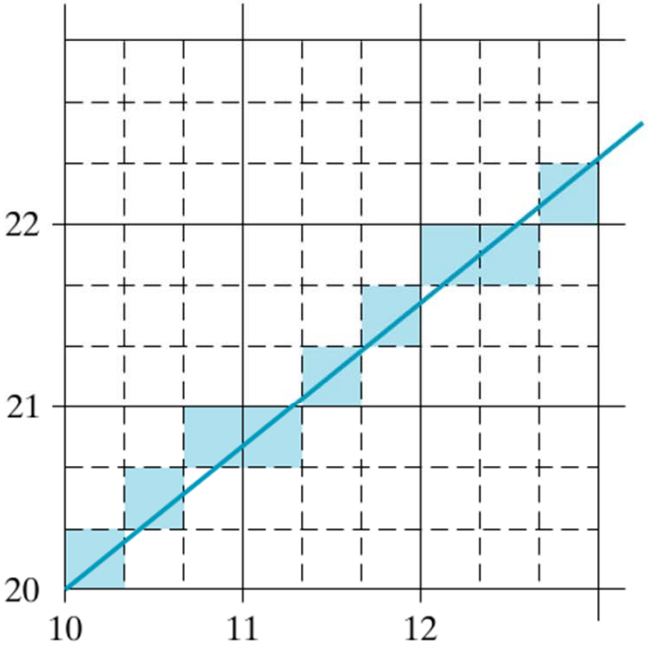


FIGURE 4-47 Supersampling subpixel positions along a straight-line segment whose left endpoint is at screen coordinates (10, 20).

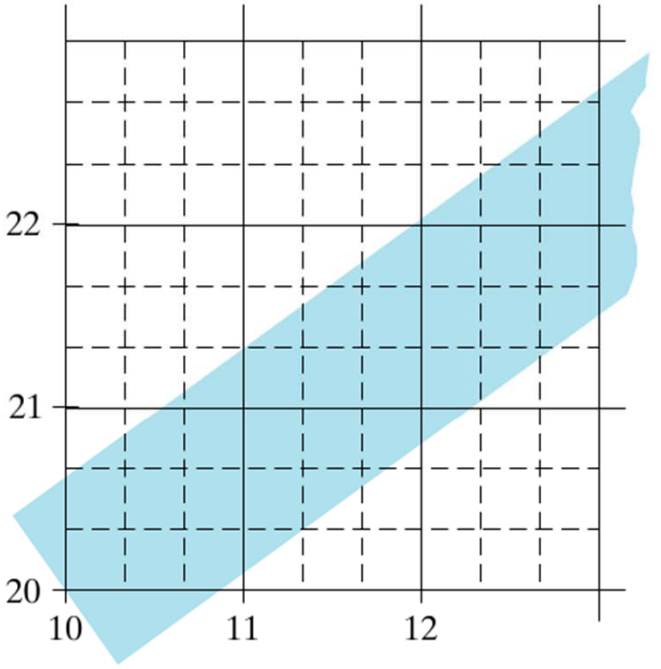


FIGURE 4-48 Supersampling subpixel positions in relation to the interior of a line of finite width.

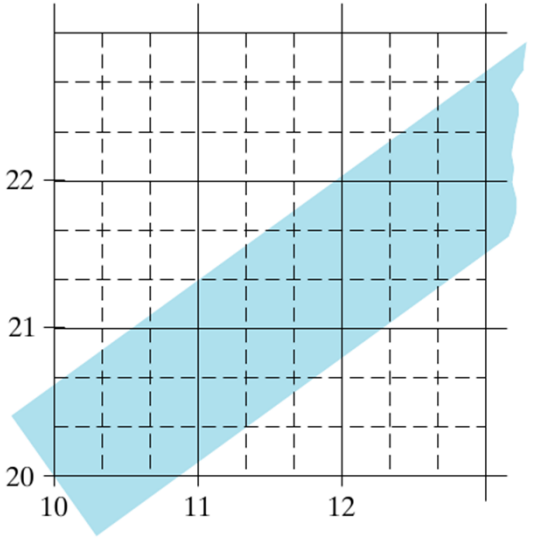
Antialiasing

- Sub-pixel Weighting Masks

1	2	1
2	4	2
1	2	1

FIGURE 4-49 Relative weights for a grid of 3 by 3 subpixels.

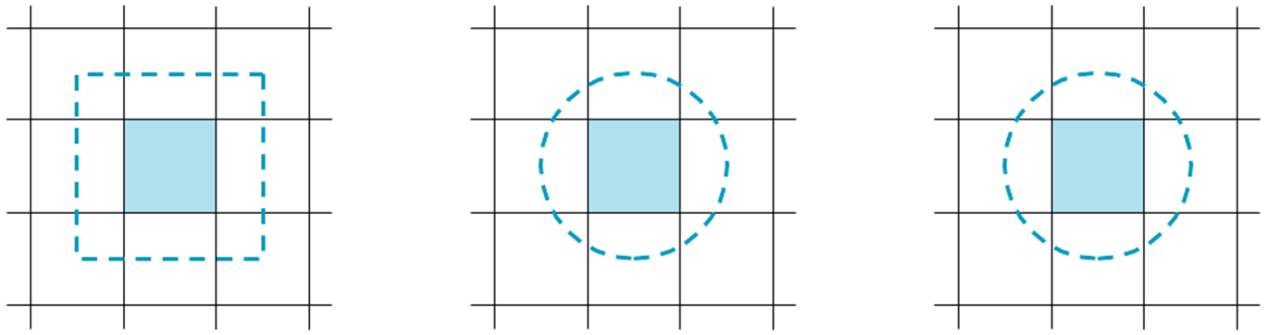
- Area Sampling Line Segments



Pixel (10,20) is about 90% covered,
Pixel (10,21) is about 15% covered

Antialiasing

- Filtering Techniques



Box Filter
(a)

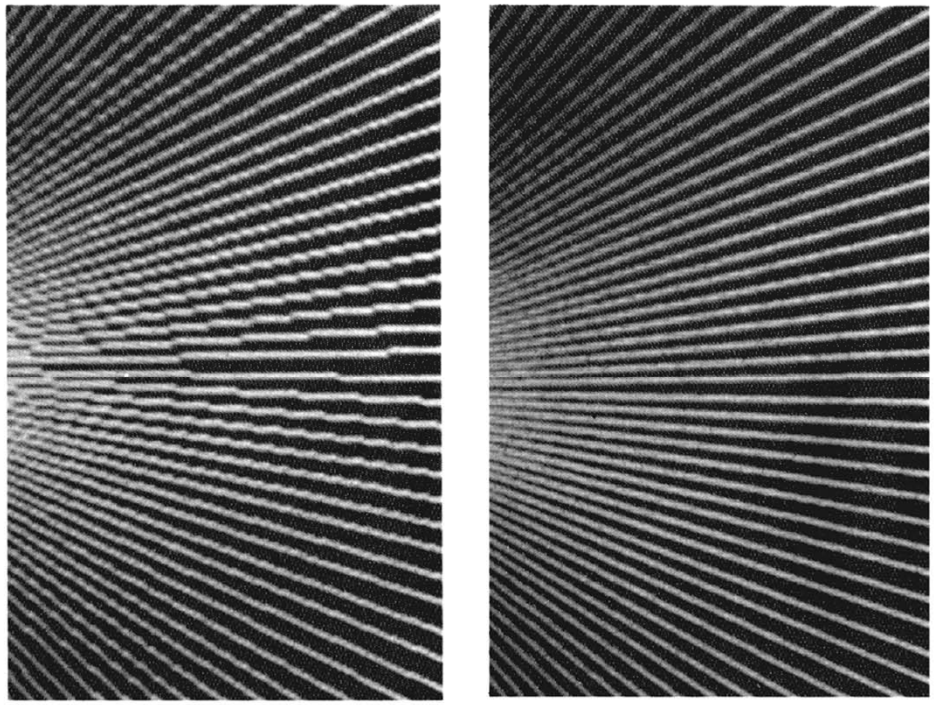
Cone Filter
(b)

Gaussian Filter
(c)

FIGURE 4–50 Common filter functions used to antialias line paths. The volume of each filter is normalized to 1.0, and the height gives the relative weight at any subpixel position.

Antialiasing

- Pixel Phasing



(a) (b)

FIGURE 4-51 Jagged lines (a), plotted on the Merlin 9200 system, are smoothed (b) with an antialiasing technique called pixel phasing. This technique increases the number of addressable points on the system from 768 by 576 to 3072 by 2304. (*Courtesy of Peritek Corp.*)

Antialiasing

- Area Boundaries

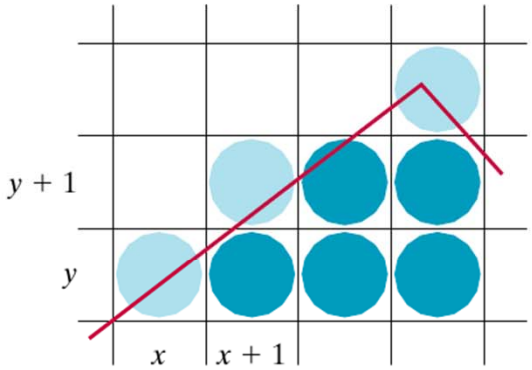


FIGURE 4–53 Adjusting pixel intensities along an area boundary.

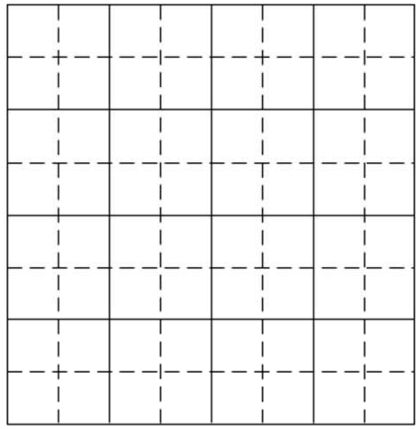


FIGURE 4–54 A 4 by 4 pixel section of a raster display subdivided into an 8 by 8 grid.

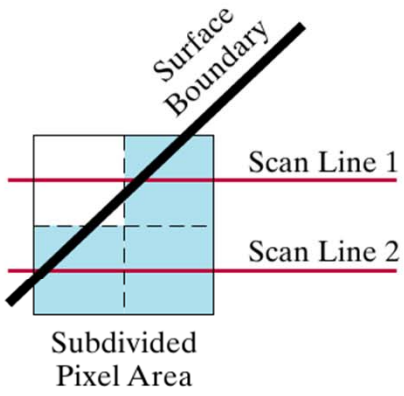


FIGURE 4–55 A subdivided pixel area with three subdivisions inside an object boundary line.

Antialiasing

- Pittway–Watkinson Algorithm

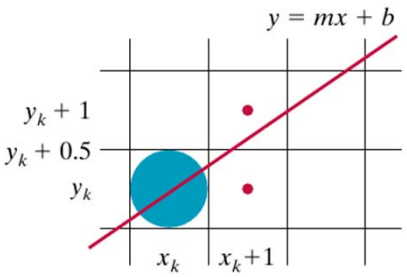


FIGURE 4-56 Boundary edge of a fill area passing through a pixel grid section.

$$y - y_{\text{mid}} = [m(x_k + 1) + b] - (y_k + 0.5) \qquad (4\text{-}14)$$

$$p = [m(x_k + 1) + b] - (y_k + 0.5) + (1 - m) \qquad (4\text{-}15)$$

Now the pixel at y_k is nearer if $p < 1 - m$, the pixel at $y_k + 1$ is nearer if $p > 1 - m$.

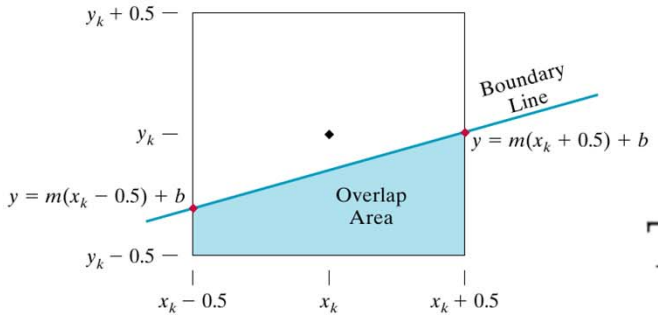


FIGURE 4-57 Overlap area of a pixel rectangle, centered at position (x_k, y_k), with the interior of a polygon fill area.

$$\text{area} = m \cdot x_k + b - y_k + 0.5 \qquad (4\text{-}16)$$

This is the same as that for p in Eq. 4-15.