Chapter 4
Attributes of Graphics Primitives

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

Attributes

- Color Attributes
- Point Attributes
- Line Attributes
- Curve Attributes
- Fill-Area Attributes
  - Fill Styles: Hollow, Solid, Patterned
- Character Attributes

RGB Color Components

TABLE 4-1

THE EIGHT RGB COLOR CODES FOR A THREE-BIT PER PIXEL FRAME BUFFER

Color Code	Stored Color Values in Frame Buffer			Displayed Color
	RED	GREEN	BLUE	
0	0	0	0	Black
1	0	0	1	Blue
2	0	1	0	Green
3	0	1	1	Cyan
4	1	0	0	Red
5	1	0	1	Magenta
6	1	1	0	Yellow
7	1	1	1	White

Color Lookup Tables

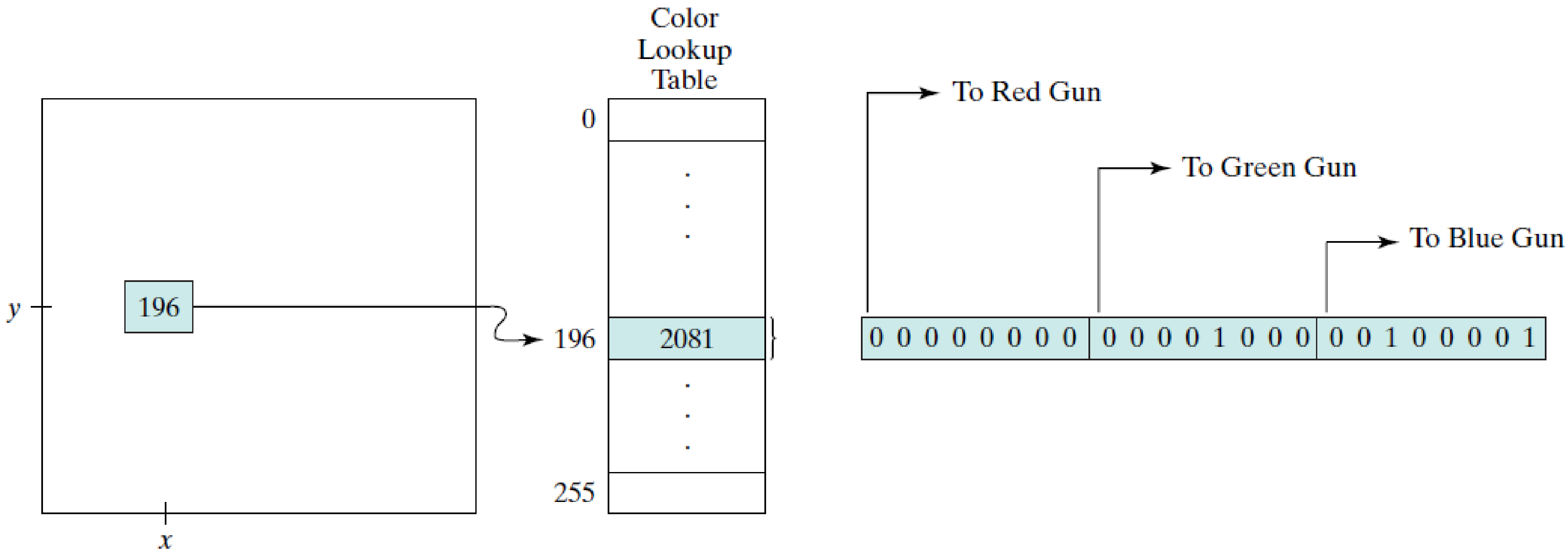


FIGURE 4-1 A color lookup table with 24 bits per entry that is accessed from a frame buffer with 8 bits per pixel. A value of 196 stored at pixel position (x, y) references the location in this table containing the hexadecimal value 0x0821 (a decimal value of 2081). Each 8-bit segment of this entry controls the intensity level of one of the three electron guns in an RGB monitor.

Color Blending

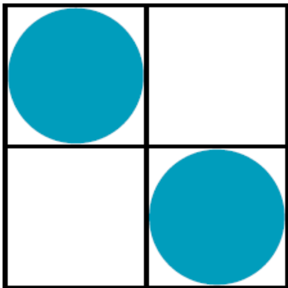
- The current color is the destination color
- The color of the second object is the source color

The new, blended color that is then loaded into the frame buffer is calculated as

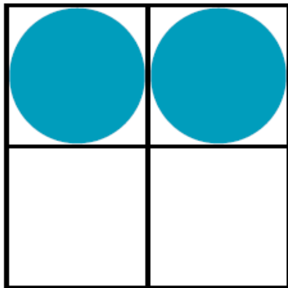
$$(S_r R_s + D_r R_d, S_g G_s + D_g G_d, S_b B_s + D_b B_d, S_a A_s + D_a A_d) \tag{4-1}$$

where the RGBA source color components are (R_s, G_s, B_s, A_s), the destination color components are (R_d, G_d, B_d, A_d), the source blending factors are (S_r, S_g, S_b, S_a), and the destination blending factors are (D_r, D_g, D_b, D_a). Computed values for the combined color components are clamped to the range from 0.0 to 1.0. That is, any sum greater than 1.0 is set to the value 1.0, and any sum less than 0.0 is set to 0.0.

Color-Blended Fill Regions

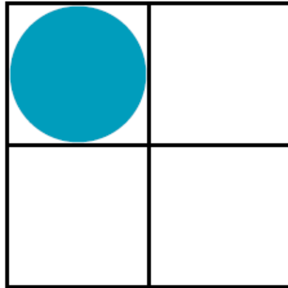


Pattern

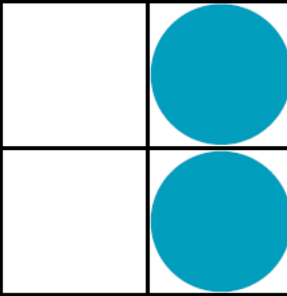


Background

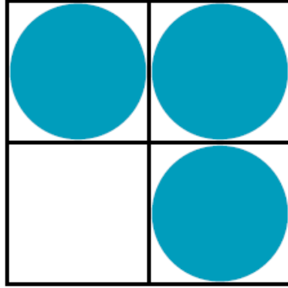
and



xor



or



replace

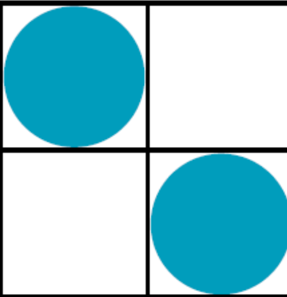


FIGURE 4–19 Combining a fill pattern with a background pattern using logical operations *and*, *or*, and *xor* (*exclusive or*), and using simple replacement.

Linear Soft-Fill Algorithm

The current RGB color **P** of each pixel within the area to be refilled is some linear combination of **F** and **B**:

$$\mathbf{P} = t\mathbf{F} + (1 - t)\mathbf{B} \tag{4-2}$$

where the transparency factor t has a value between 0 and 1 for each pixel. For values of t less than 0.5, the background color contributes more to the interior color of the region than does the fill color. Vector Eq. 4-2 holds for each RGB component of the colors, with

$$\mathbf{P} = (P_R, P_G, P_B), \qquad \mathbf{F} = (F_R, F_G, F_B), \qquad \mathbf{B} = (B_R, B_G, B_B) \tag{4-3}$$

Similar color-blending procedures can be applied to an area whose foreground color is to be merged with multiple background color areas, such as a checkerboard pattern. When two background colors B_1 and B_2 are mixed with foreground color **F**, the resulting pixel color **P** is

$$\mathbf{P} = t_0\mathbf{F} + t_1\mathbf{B_1} + (1 - t_0 - t_1)\mathbf{B_2} \tag{4-5}$$

Scan-Line Polygon Fill

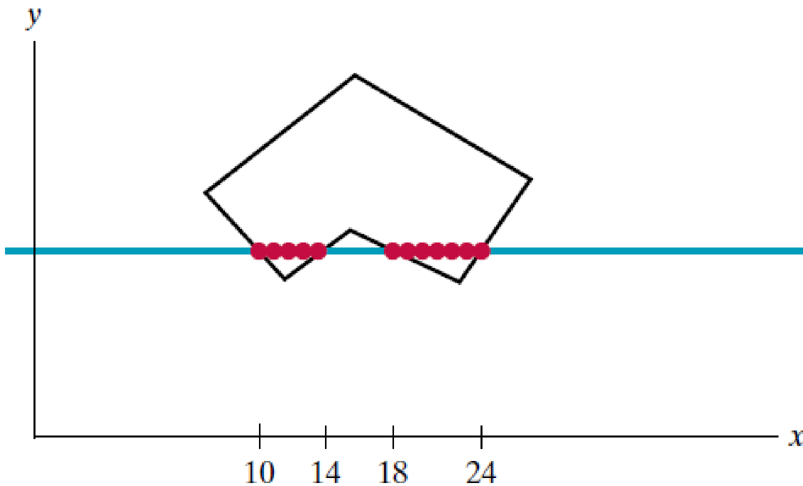


FIGURE 4-20 Interior pixels along a scan line passing through a polygon fill area.

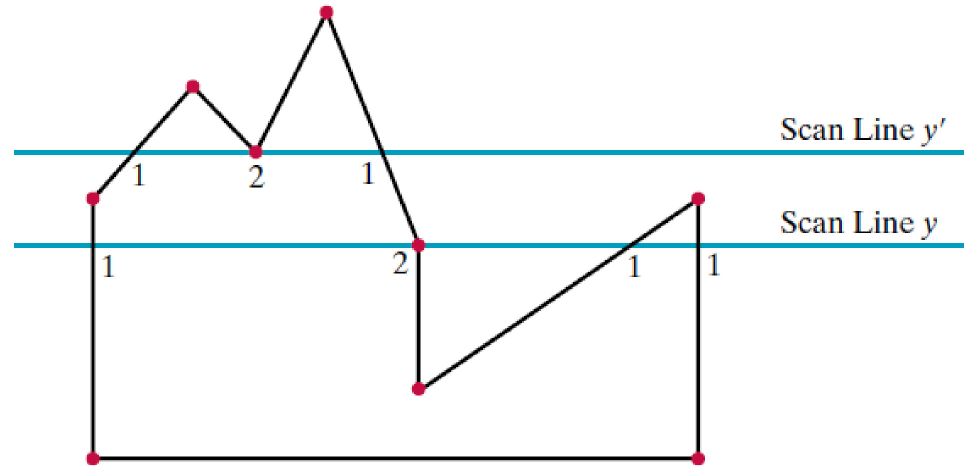


FIGURE 4-21 Intersection points along scan lines that intersect polygon vertices. Scan line y generates an odd number of intersections, but scan line y' generates an even number of intersections that can be paired to identify correctly the interior pixel spans.

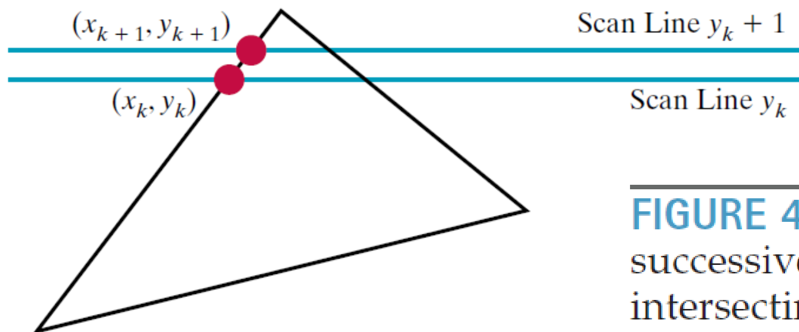


FIGURE 4-23 Two successive scan lines intersecting a polygon boundary.

Scan-Line Polygon Fill

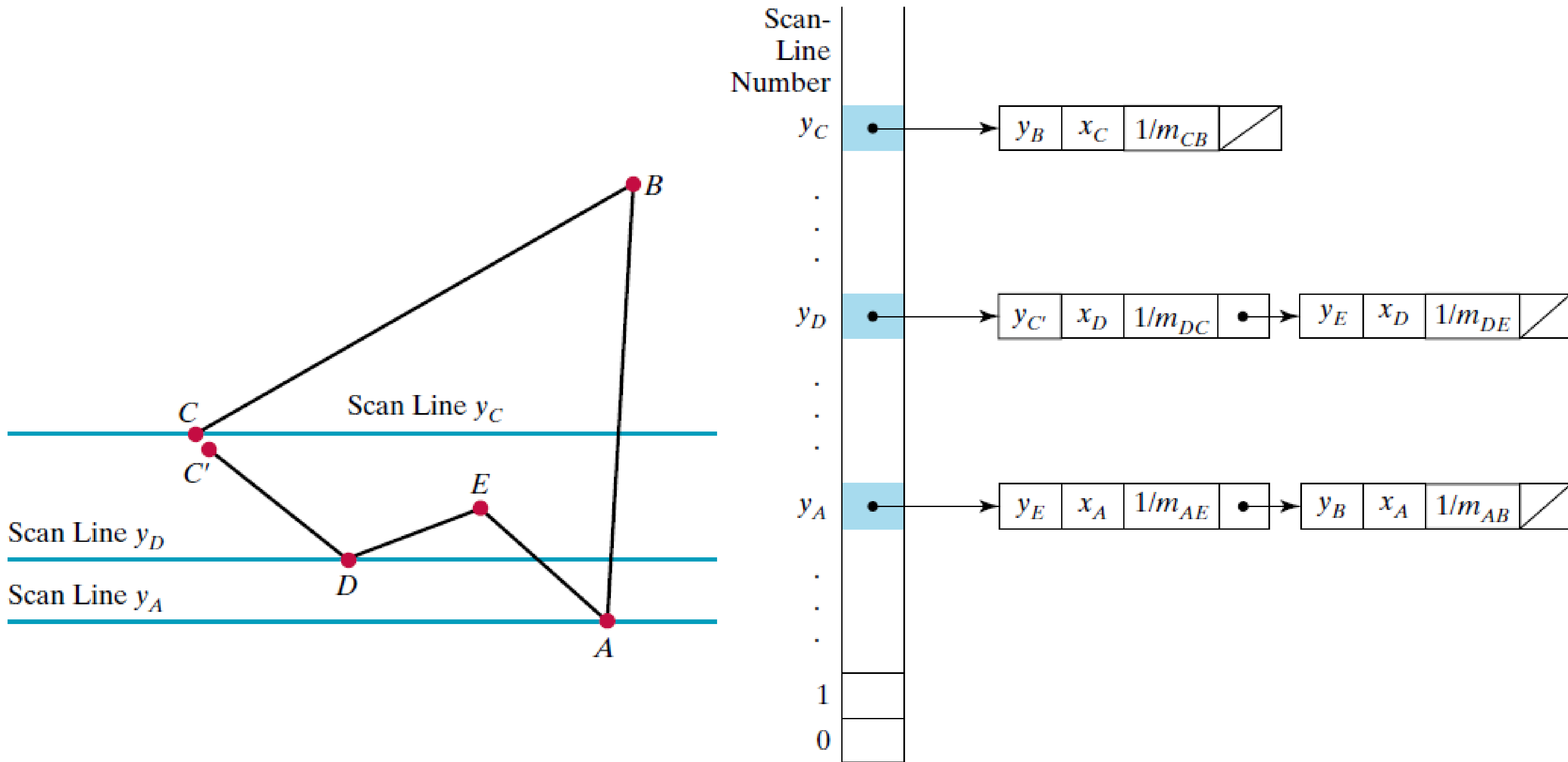


FIGURE 4-24 A polygon and its sorted edge table, with edge $\overline{DC}$ shortened by one unit in the y direction.

Boundary-Fill Algorithm

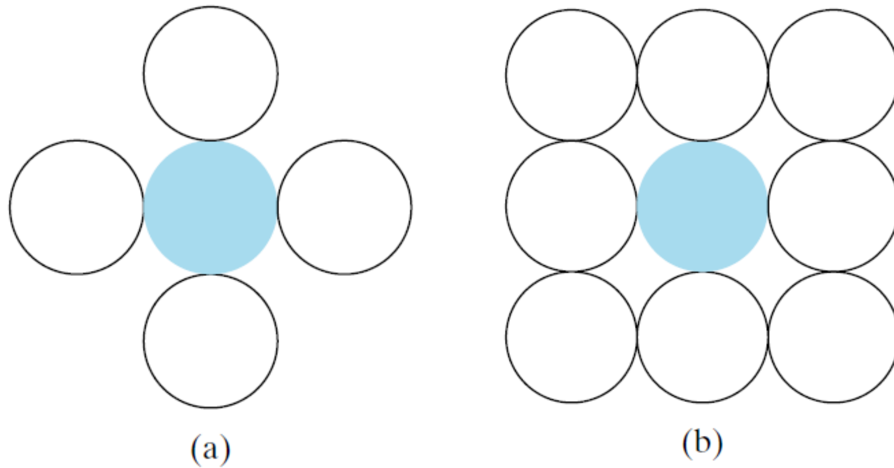


FIGURE 4–27 Fill methods applied to a 4-connected area (a) and to an 8-connected area (b). Hollow circles represent pixels to be tested from the current test position, shown as a solid color.

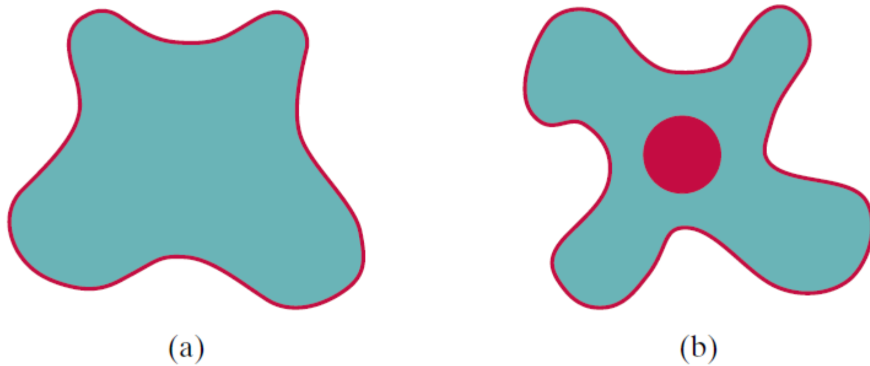


FIGURE 4–26 Example color boundaries for a boundary-fill procedure.

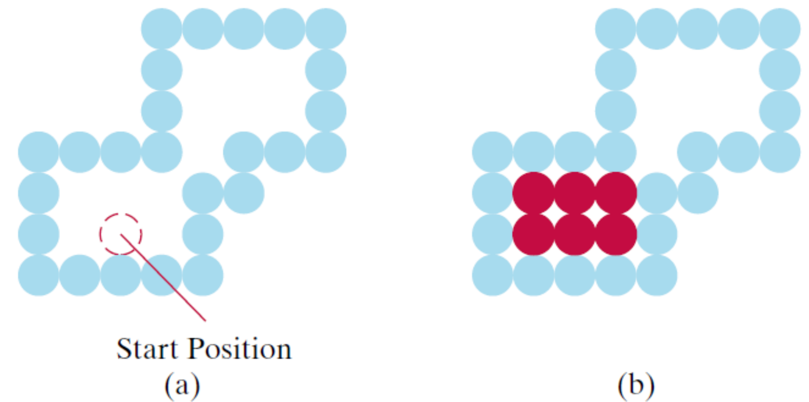


FIGURE 4–28 The area defined within the color boundary (a) is only partially filled in (b) using a 4-connected boundary-fill algorithm.

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

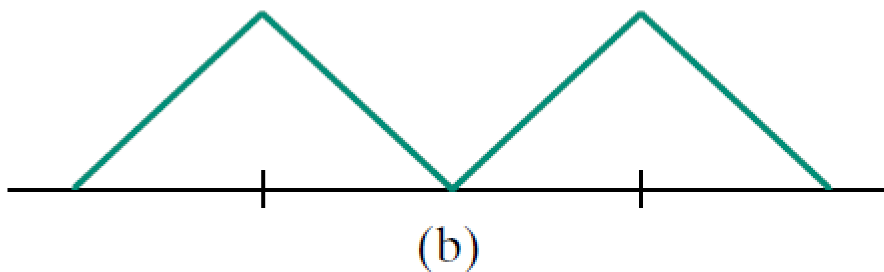


FIGURE 4-46 Sampling the periodic shape in (a) at the indicated positions produces the aliased lower-frequency representation in (b).

Antialiasing

- Super-sampling Straight-Line Segments

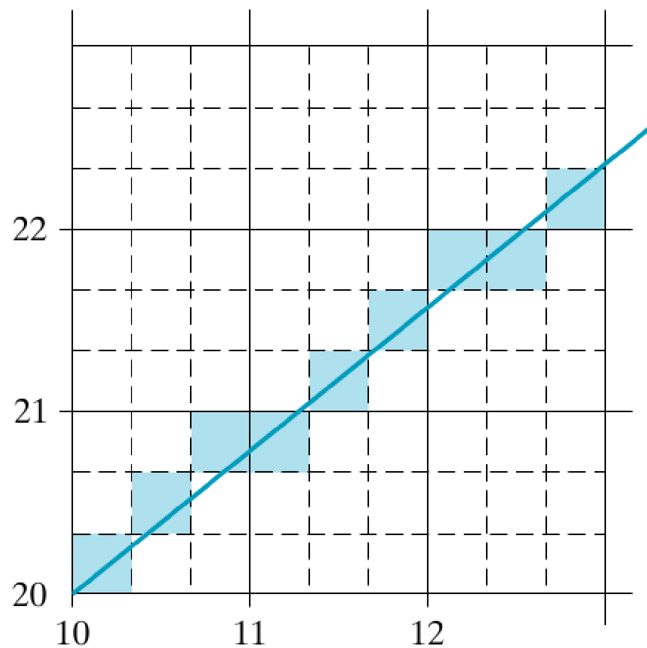


FIGURE 4-47 Supersampling subpixel positions along a straight-line segment whose left endpoint is at screen coordinates (10, 20).

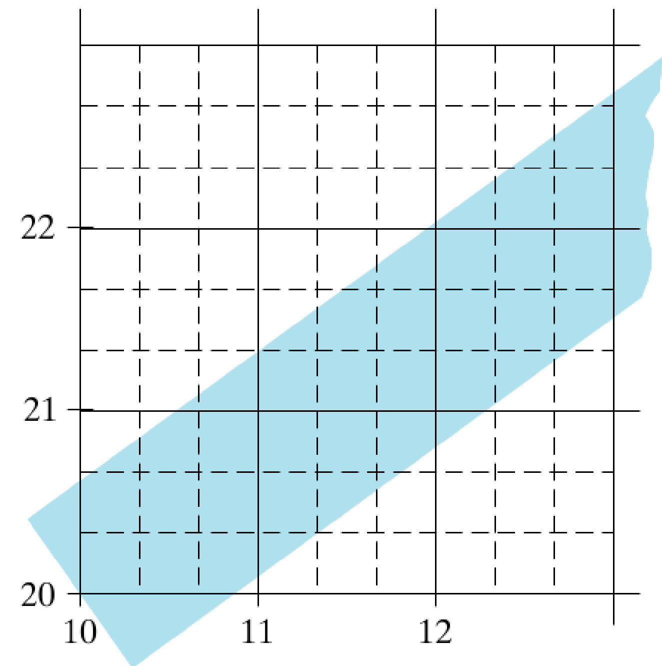


FIGURE 4-48 Supersampling subpixel positions in relation to the interior of a line of finite width.

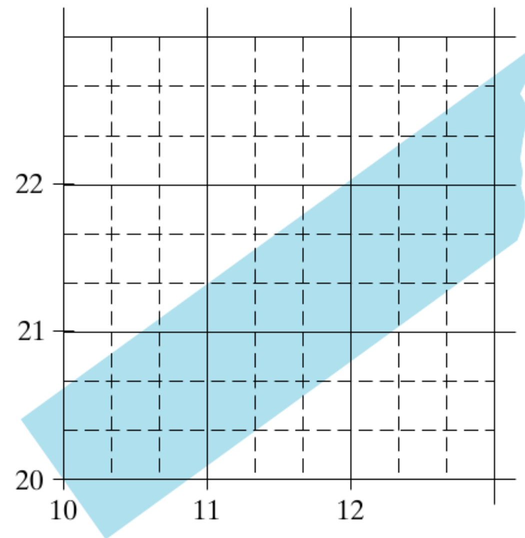
Antialiasing

- 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.

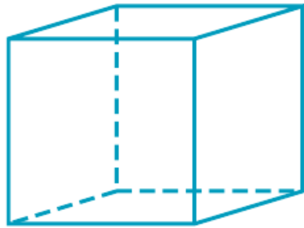
- Area Sampling Line Segments

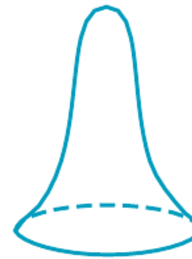


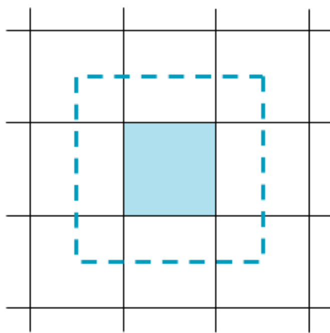
Pixel (10,20) is about 90% covered,
Pixel (10,21) is about 15% covered

Antialiasing

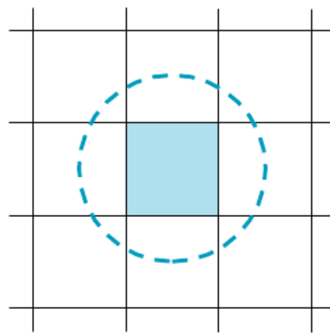
- Filtering Techniques



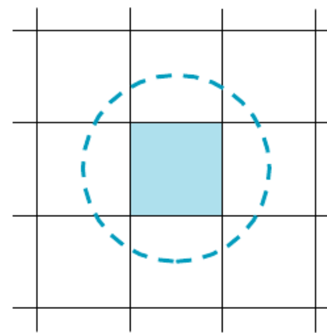




Box Filter
(a)



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

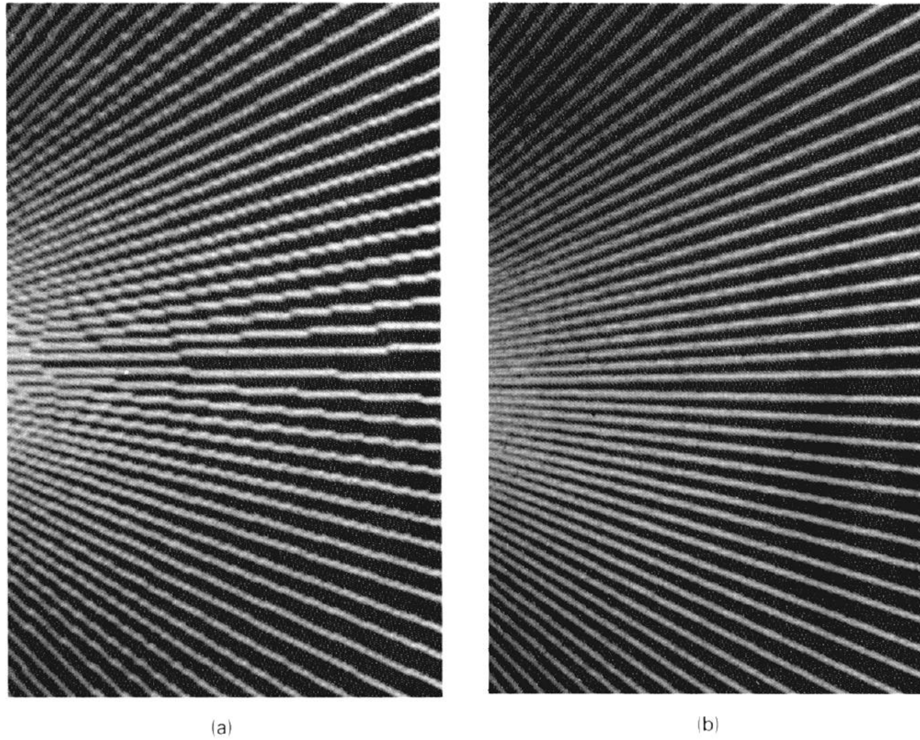


FIGURE 4-51 Jagged lines (a), plotted on the Merlin 9200 system, are smoothed (b) with an antialiasing technique called pixel phasing. This technique increases the number of addressable points on the system from 768 by 576 to 3072 by 2304. (*Courtesy of Peritek Corp.*)

Antialiasing

- Area Boundaries

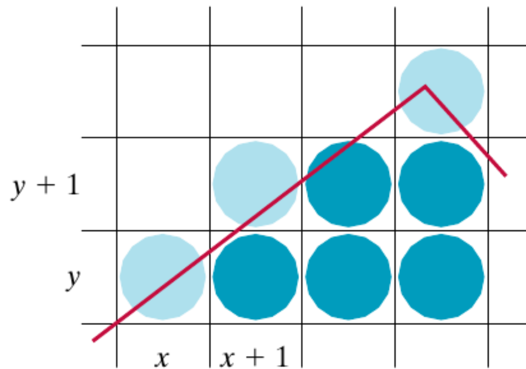


FIGURE 4-53 Adjusting pixel intensities along an area boundary.

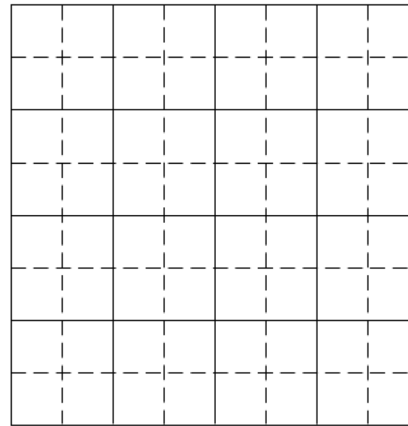


FIGURE 4-54 A 4 by 4 pixel section of a raster display subdivided into an 8 by 8 grid.

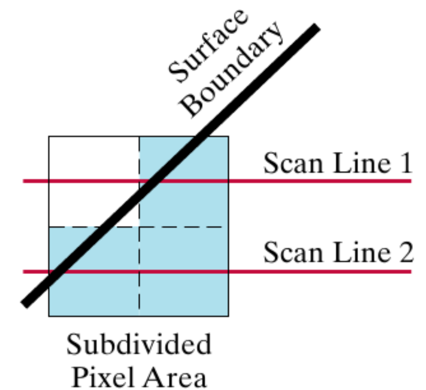


FIGURE 4-55 A subdivided pixel area with three subdivisions inside an object boundary line.

Antialiasing

- Pittway–Watkinson Algorithm

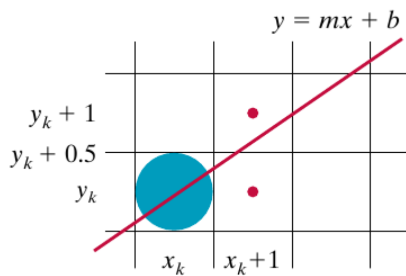


FIGURE 4–56 Boundary edge of a fill area passing through a pixel grid section.

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

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

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

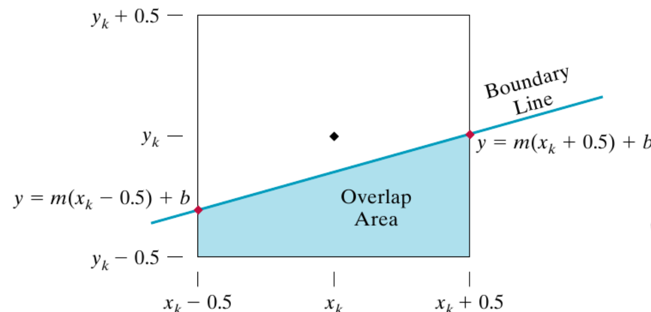


FIGURE 4–57 Overlap area of a pixel rectangle, centered at position (x_k, y_k), with the interior of a polygon fill area.

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

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