COMPUTER GRAPHICS

Attributes of Graphics Primitives

Hearn & Baker Chapter 4

Some slides are taken from Robert Thomsons notes.

OVERVIEW

- Color
- Point and line attributes
- Curve attributes
- Fill-area attributes
- Scan-line fill
- Fill-area methods
- Character attributes
- Antialising

Attributes

- Attribute parameter is a parameter that affects the way a primitive is to be displayed.
- Some attribute parameters determine the fundamental characteristics of a primitive.
  - Color
  - Size
- For now, we consider only those attributes that control the basic display properties of primitives.
  - Lines can be dotted or dashed, fat or thin, and blue or orange.
  - Areas might be filled with one color or with a multicolor pattern.

Color

- Basic attribute for all primitives.
- RGB color components
  - Either store the color code into frame buffer
  - Or use index values to reference the color-table entries.

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

Color	Stored Color Values in Frame Buffer			Displayed Color
Code	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 tables

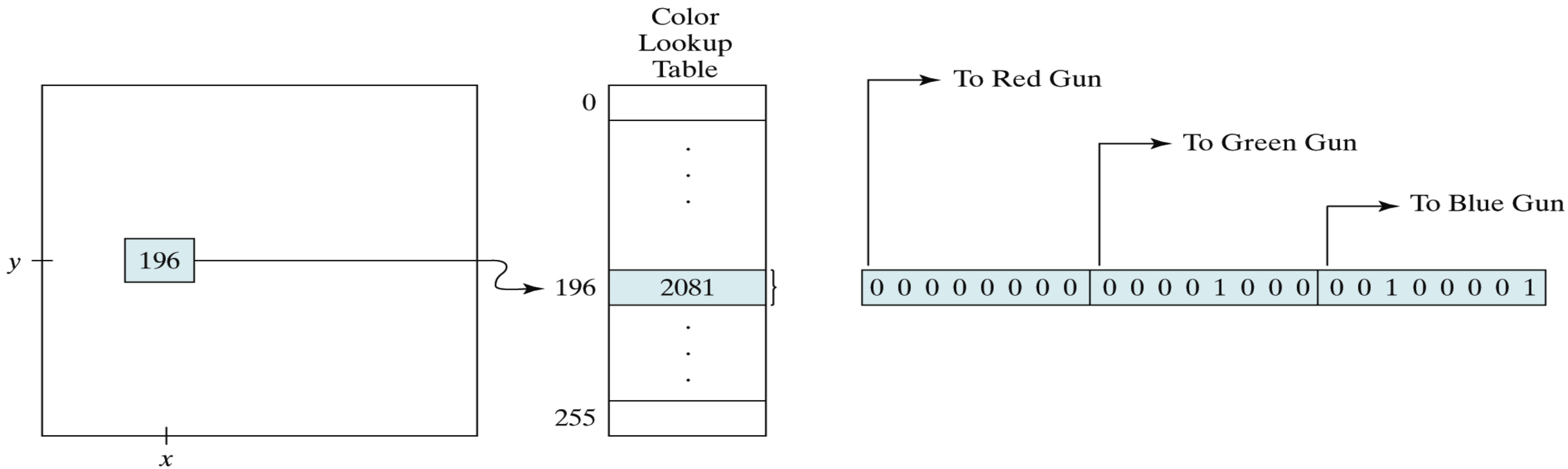


Figure 4-1

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.

Line attributes

- Line width
  - Plot a vertical span of pixels in each column (slope <=1)
  - Line width is equal to number of pixels in each column

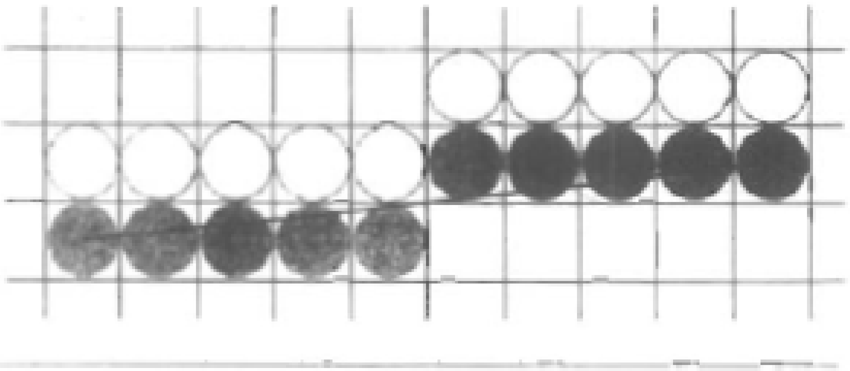


Figure 4-3
Double-wide raster line with slope $|m| < 1$ generated with vertical pixel spans.

Line width

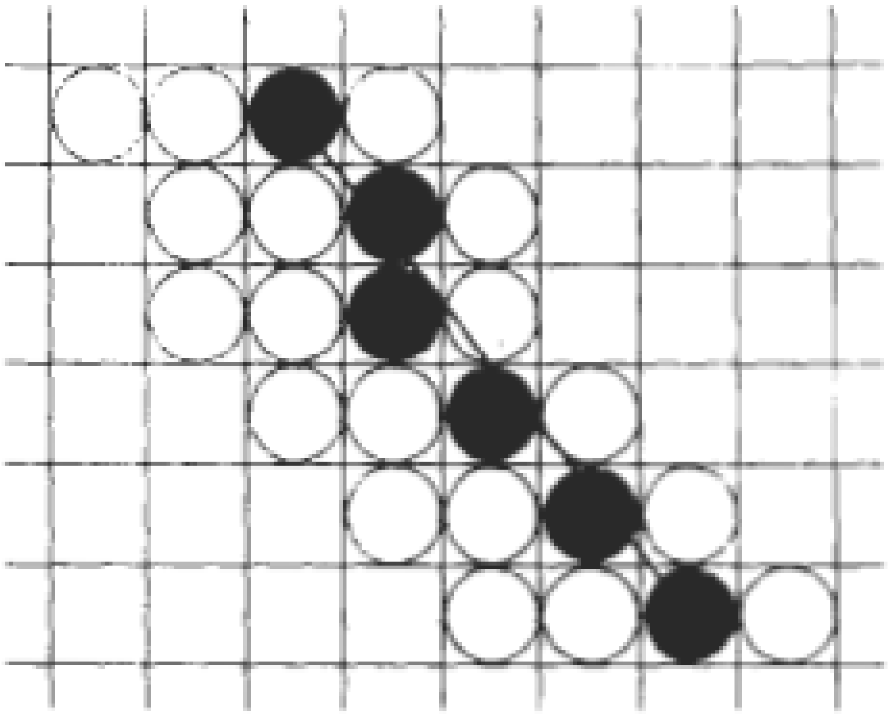
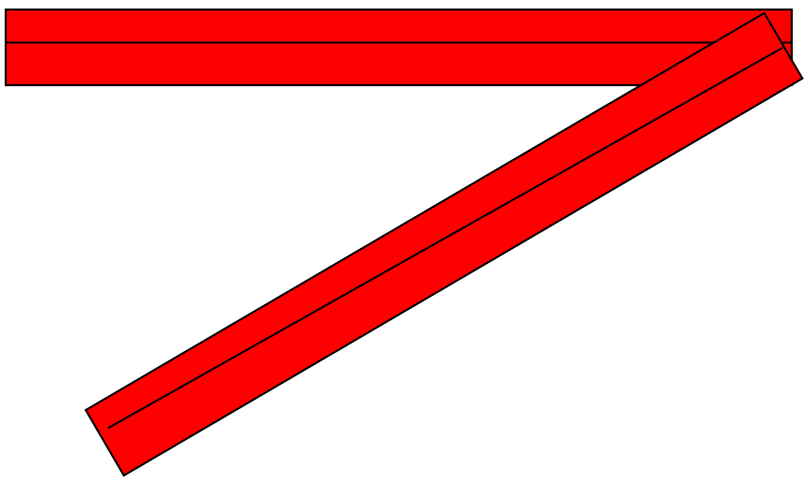


Figure 4-4
Raster line with slope $|m| > 1$
and line-width parameter lw = 4
plotted with horizontal pixel spans.

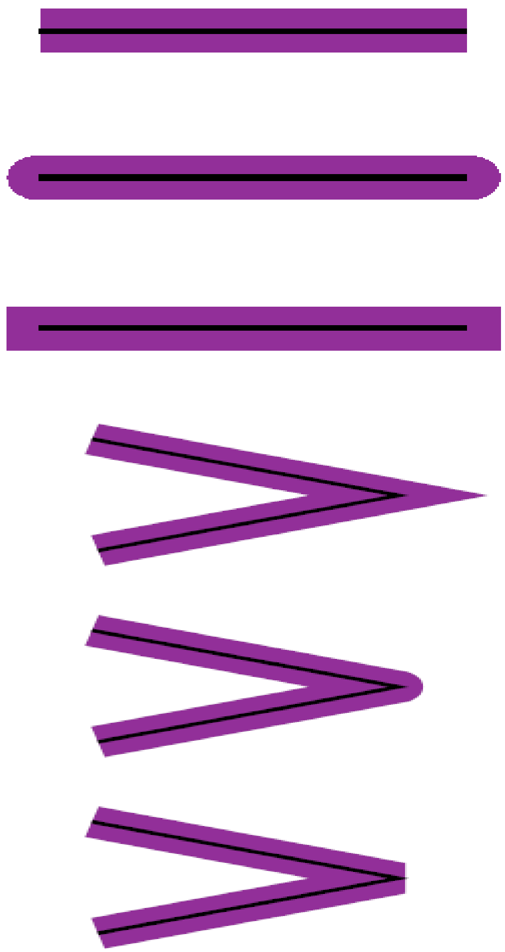
- Problem
  - Width of a line depends on the slope. 45^0 line will be thinner by a factor of 1/sqrt(2)

Line width

- Another problem: line ends
  - Horizontal or vertical line ends regardless of slope
- Solution
  - Add line caps



Line Attributes



Butt cap

Round cap

Projecting square cap

Miter join

Round Join

Bevel join

Pen and Brush Options

- Using packages, lines can be displayed with pen or brush selections.
  - shape, size, and pattern.
- Some possible pen or brush shapes

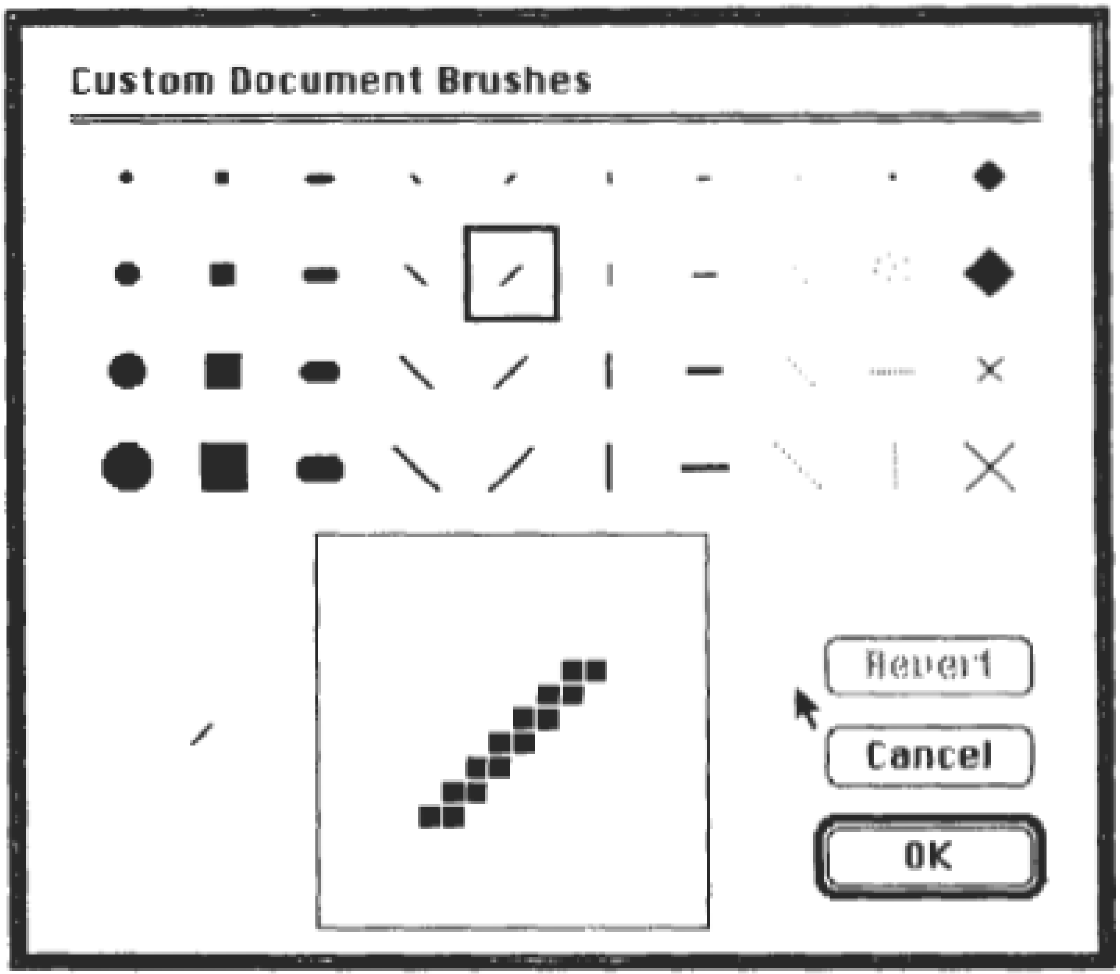


Figure 4-7
Pen and brush shapes for line display.

- The shapes can be stored in a pixel mask
  - I.e., a rectangular pen can be implemented with the following mask

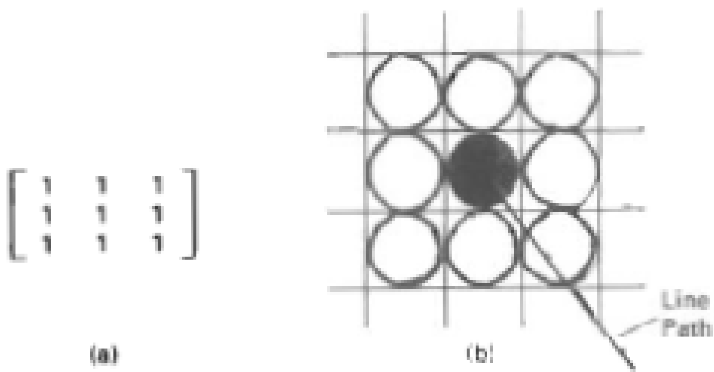


Figure 4-8
(a) A pixel mask for a rectangular pen, and (b) the associated array of pixels displayed by centering the mask over a specified pixel position.

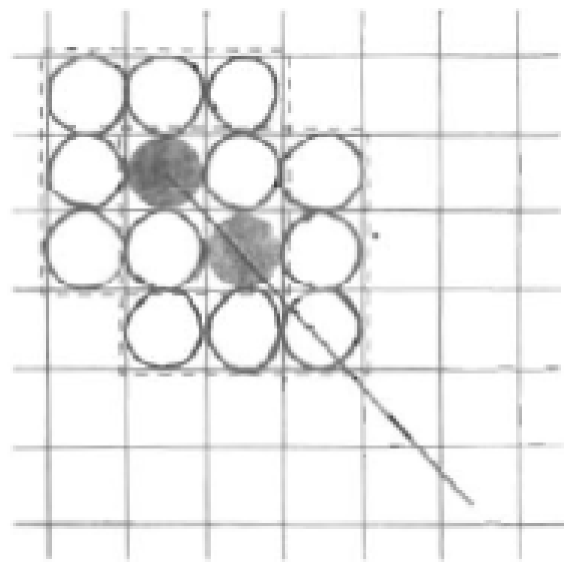
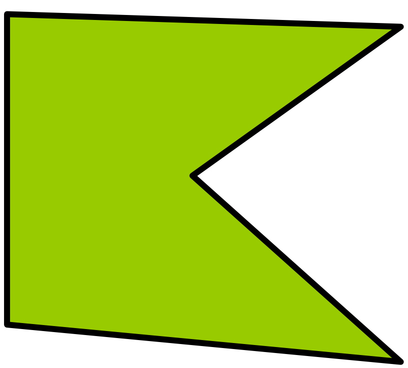
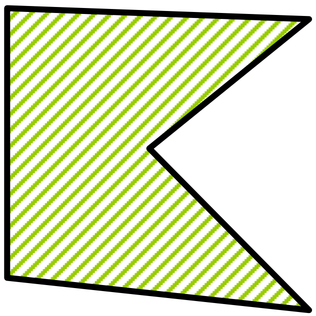


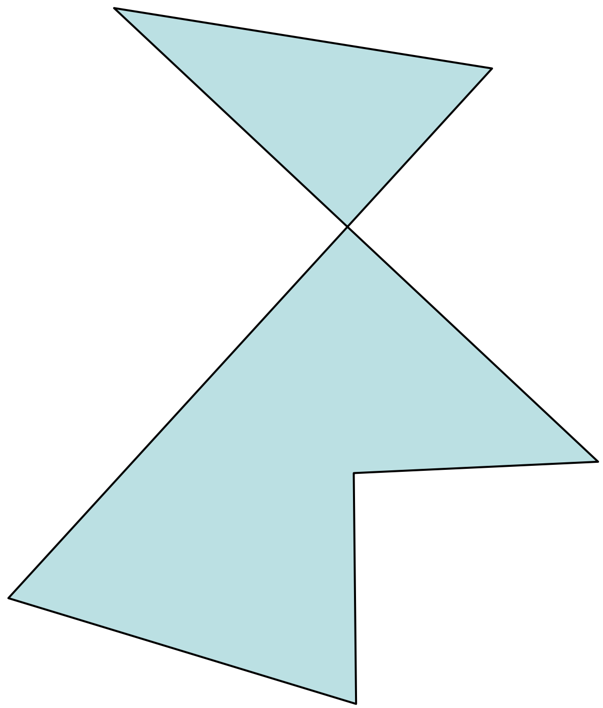
Figure 4-9
Generating a line with the pen shape of Fig. 4-8.

Area Filling

How can we generate a solid color/patterned polygon area?

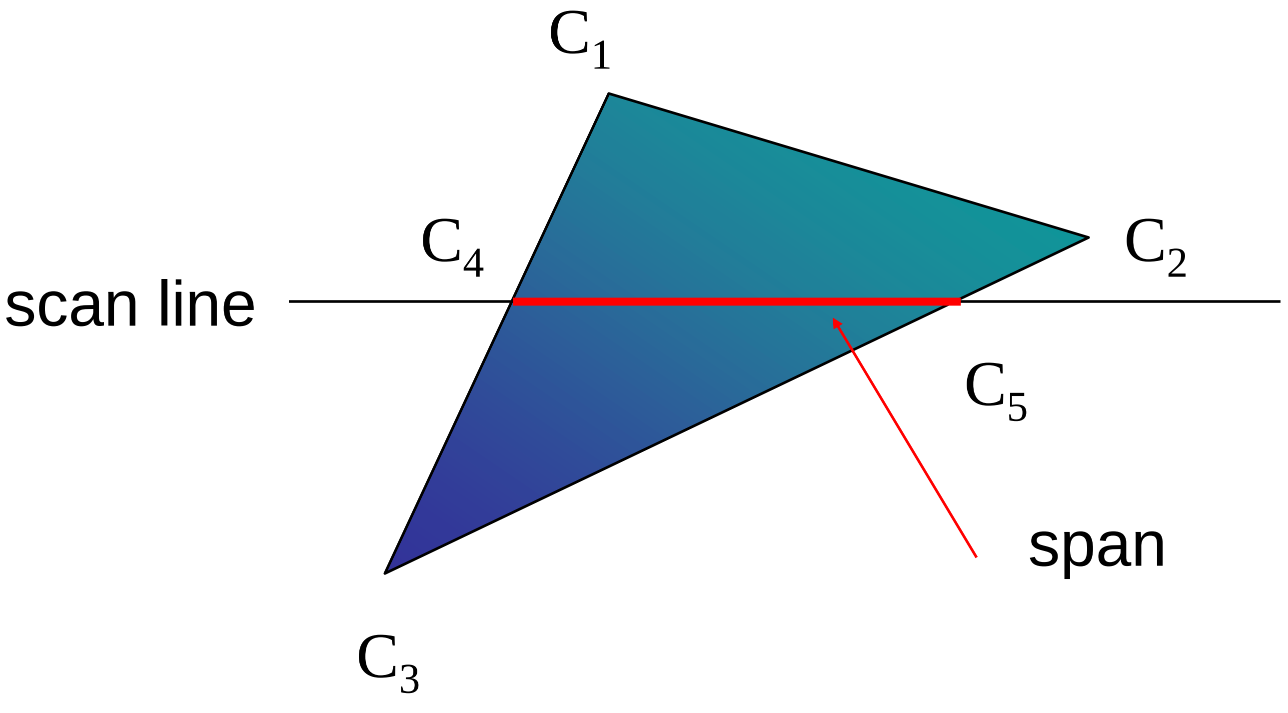






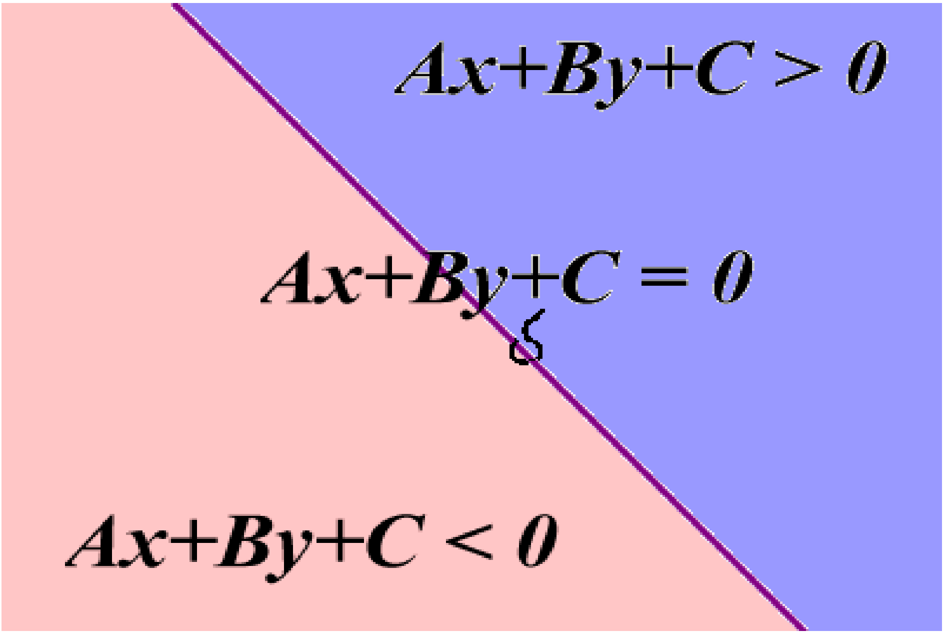
Triangle fill : color Interpolation

C_1 C_2 C_3 specified by **`glColor`** or by vertex shading
C_4 determined by interpolating between C_1 and C_3
C_5 determined by interpolating between C_2 and C_3
interpolate between C_4 and C_5 along span



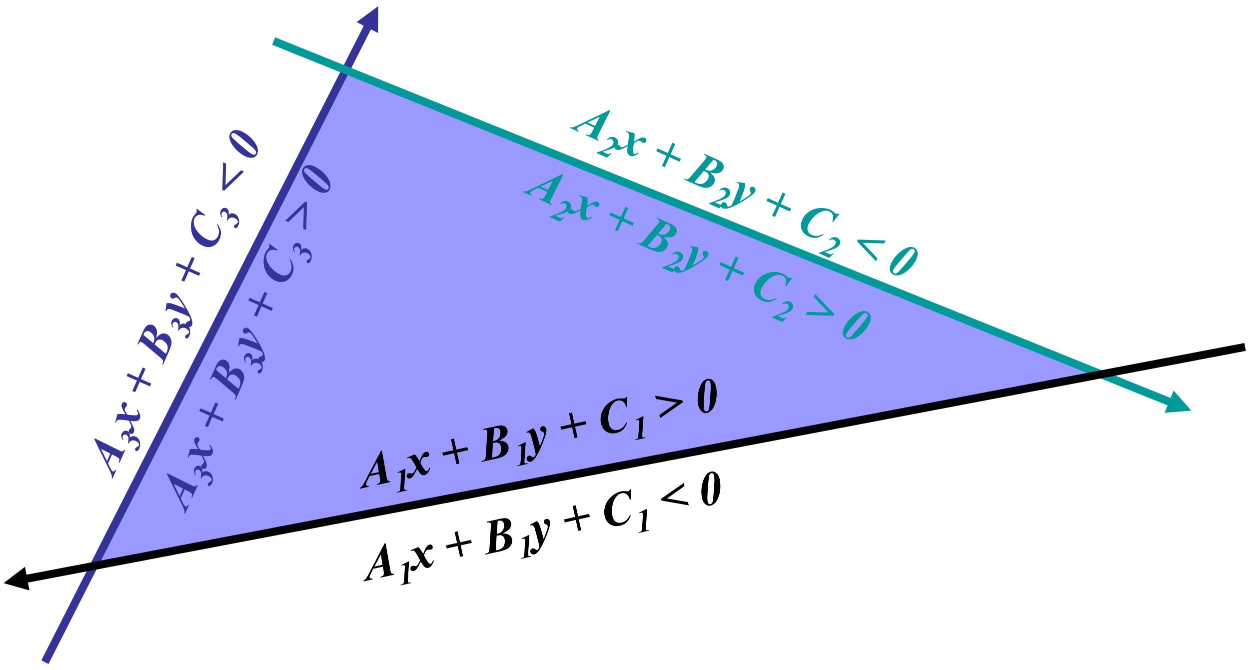
Edge equations

- An edge equations is the equations of the line defining the edges
  - Each line defines 2 half-spaces



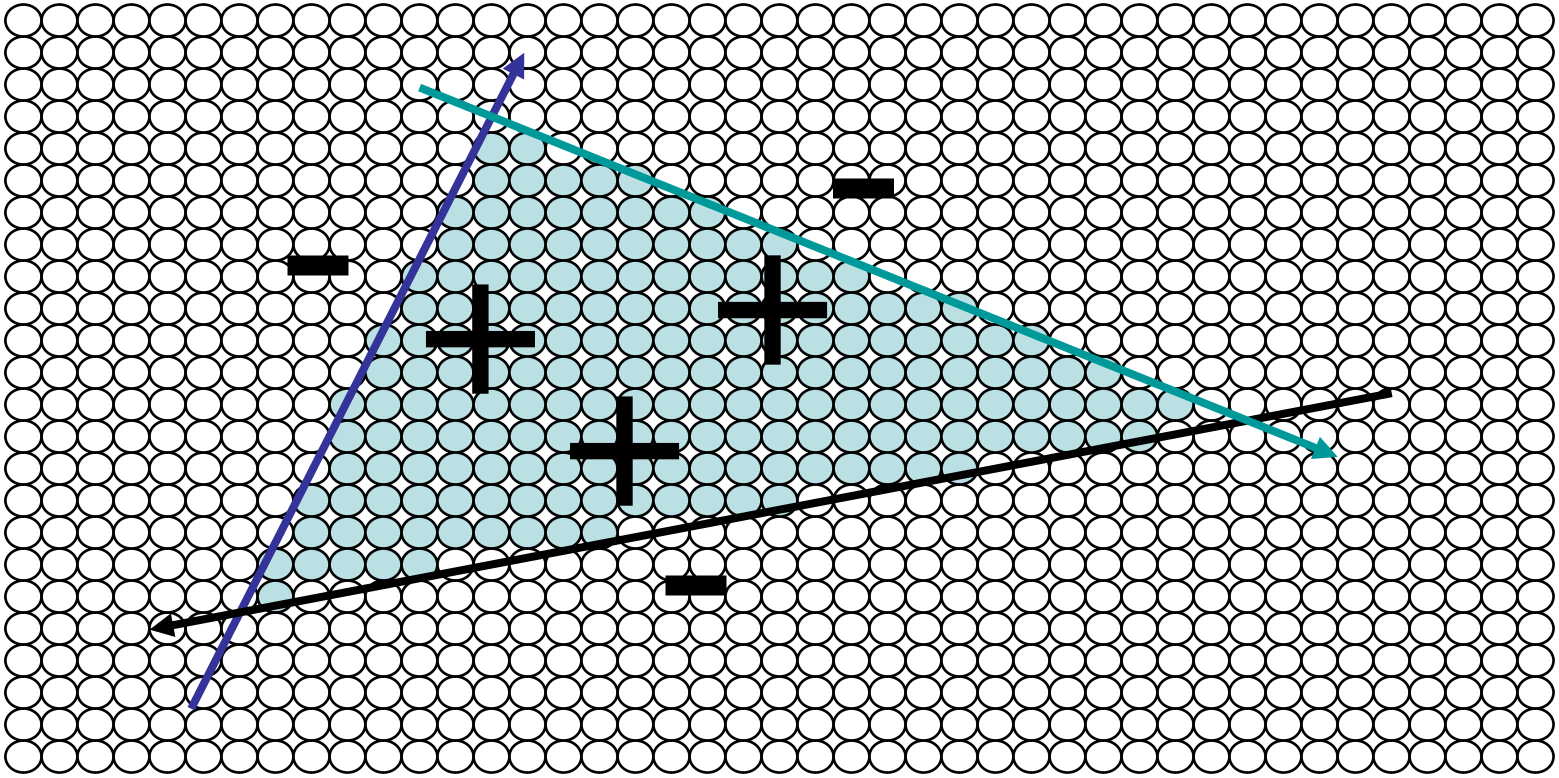
Edge equations

- A triangle can be defined as the intersection of three positive half-spaces



Edge Equations

So…simply turn on those pixels for which all edge equations evaluate to > 0:



Lots of implementation details to consider….

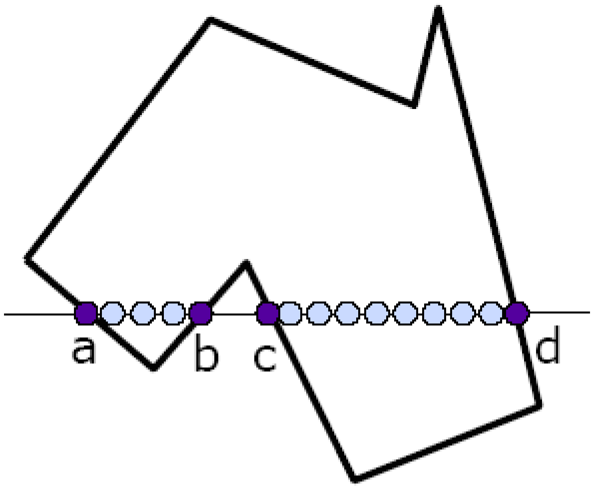
Filled Area Primitives

- Two basic approaches to area filling on raster systems:
  - Determine the overlap intervals for scan lines that cross the area (scan-line)
  - Start from an interior position and point outward from this point until the boundary condition reached (fill method)
- Scan-line: simple objects, polygons, circles,..
- Fill-method: complex objects, interactive fill.

Scan-line Polygon Fill

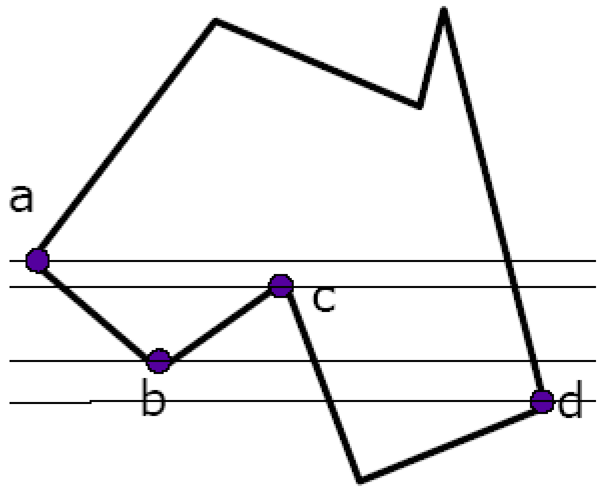
(Line Walking approach)

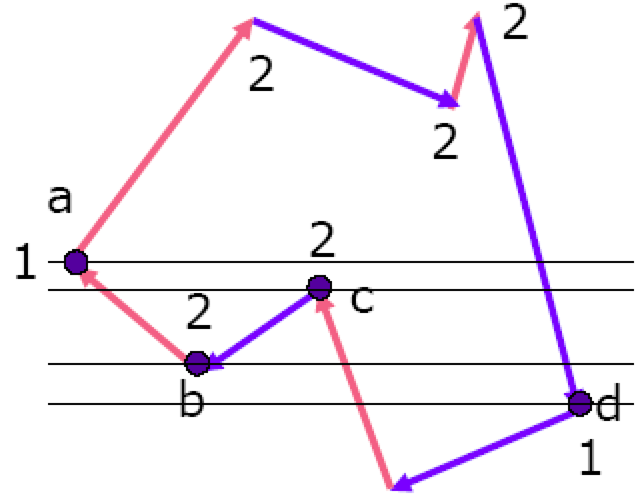
- For each scan-line:
  - Locate the intersection of the scan-line with the edges ($y=y_s$)
  - Sort the intersection points from left to right.
  - Draw the interiors intersection points pairwise. (a-b), (c-d)
- Problem with corners. Same point counted twice or not?



(When deciding if scan-line is inside the polygon)

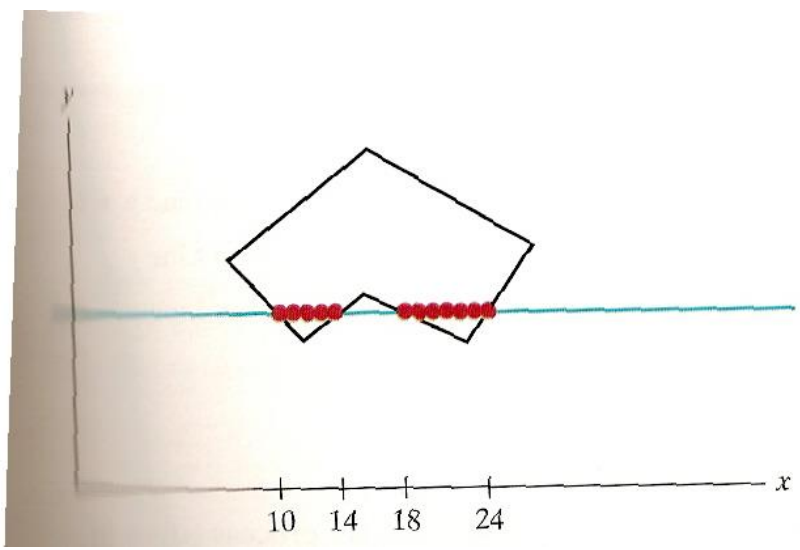
- *a*,*b*,*c* and *d* are intersected by 2 line segments each.
- Count *b*,*c* twice but *a* and *d* once. Why?
- Solution:
  Make a clockwise or counter-clockwise traversal on edges.
  Check if y is monotonically increasing or decreasing.
  If direction changes, double intersection, otherwise single intersection.

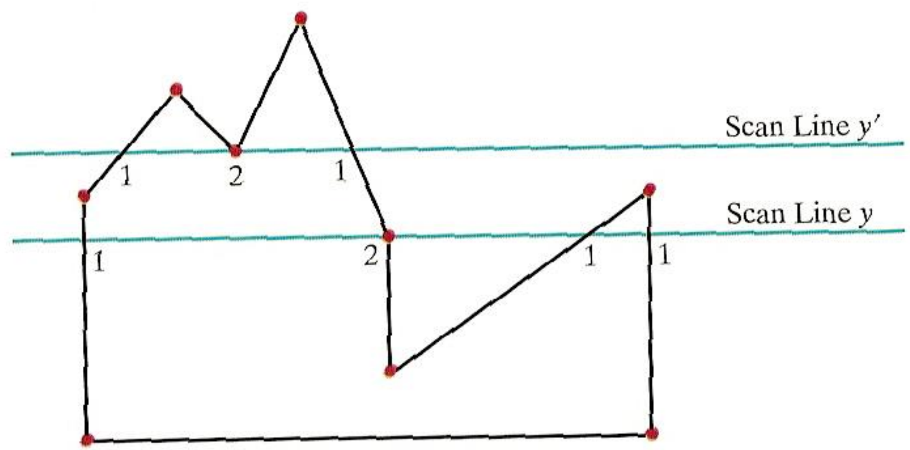




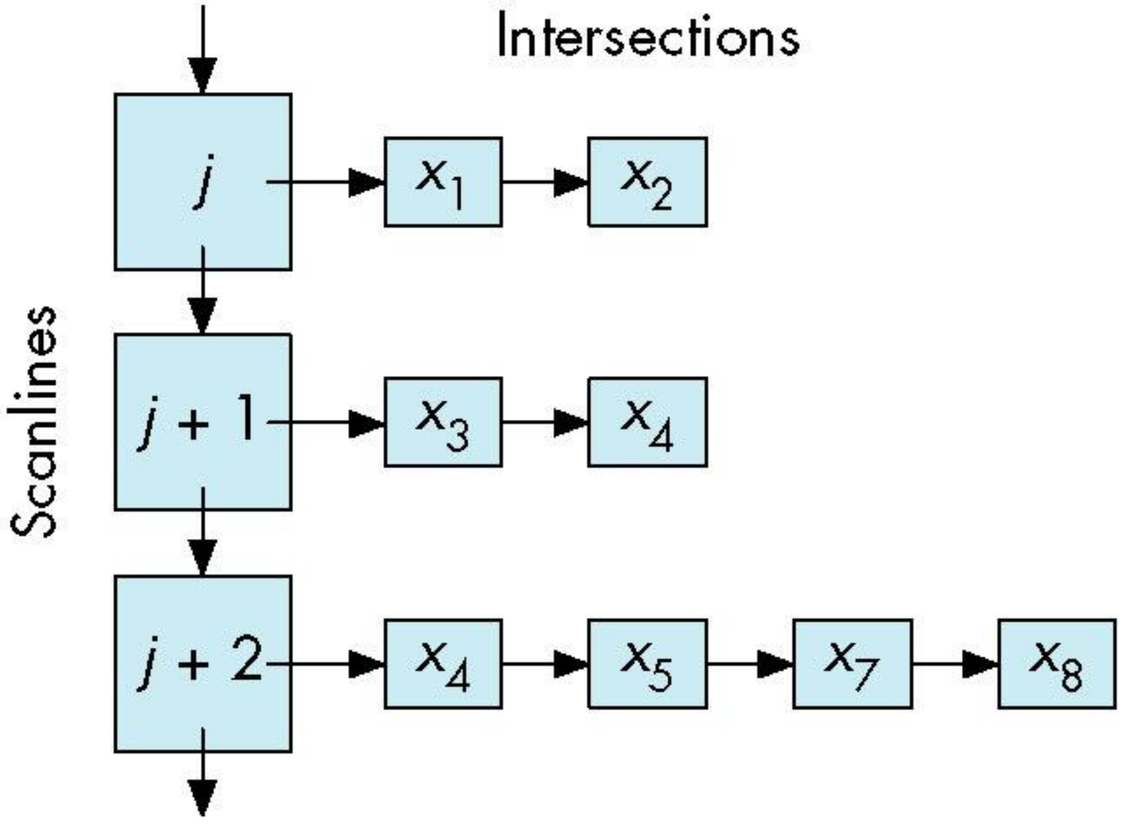
Area Filling (Scan line Approach)

- For each scan line (increasing y order)
  (1) Find intersections (the extrema of spans)
  (2) Sort intersections (increasing x order)
  (3) Fill in between pair of intersections





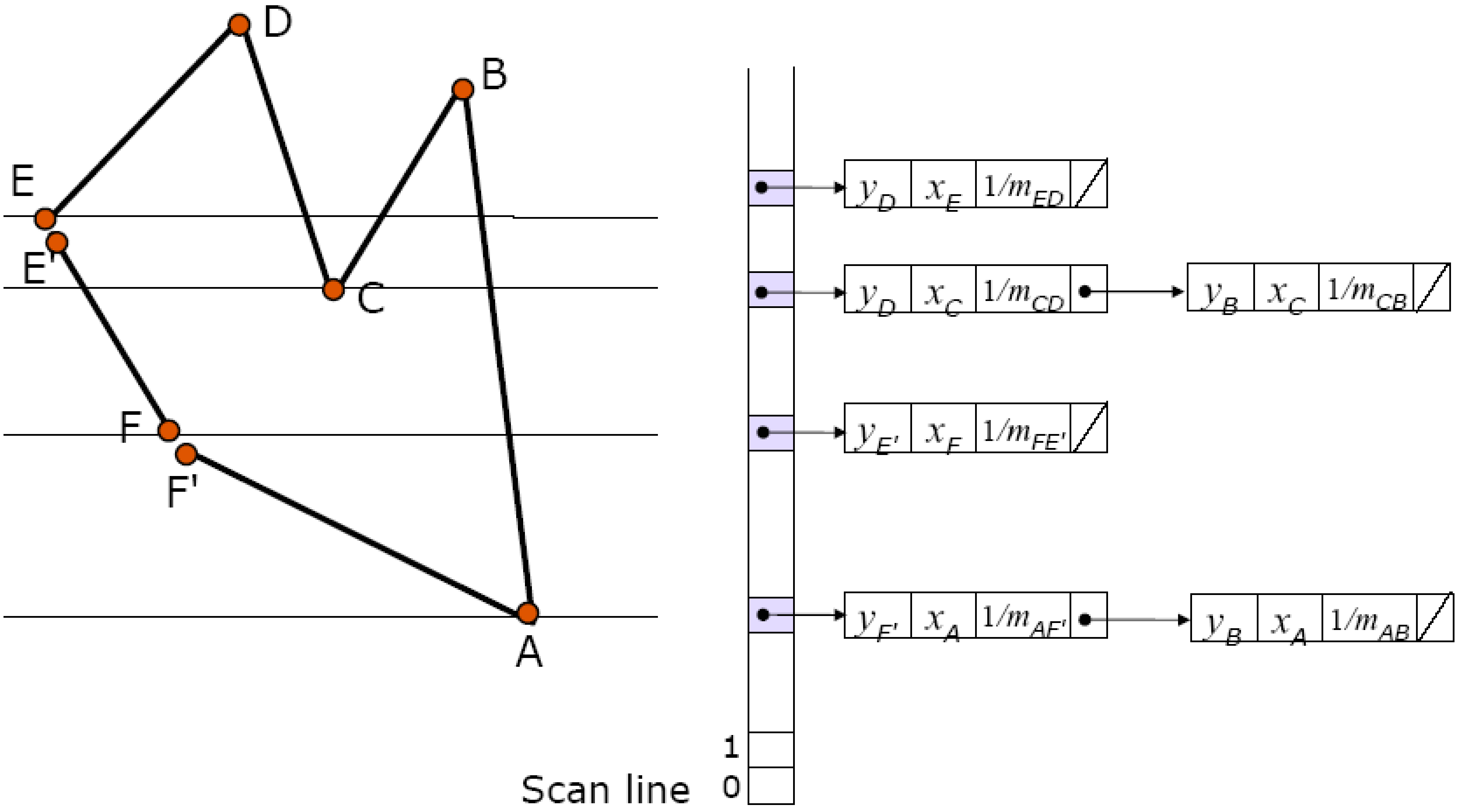
Data Structure



Efficient Polygon Fill

- Make a (counter) clockwise traversal and shorten the single intersection edges by one pixel (so that we do not need to re-consider single/double edges).
- Generate a sorted edge table on the scan-line axis. Each edge has an entry in smaller y valued corner point (vertex).
- Each entry keeps a linked list of all connected edges:
  - x value of the point
  - y value of the end-point
  - Slope of the edge

Sorted edge table

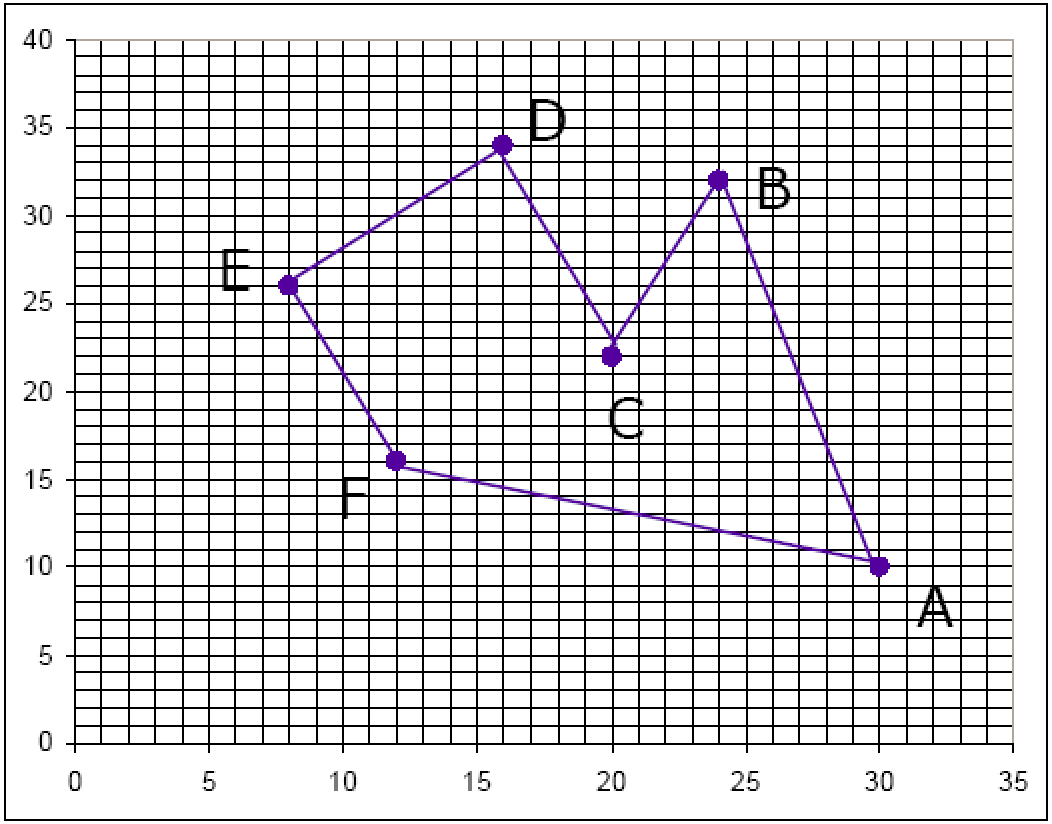


- Start with the smallest scan-line
- Keep an **active edge list**:
  - Update the current *x* value of the edge based on *m* value
  - Add the lists in the current table entry based on their *x* value
  - Remove the completed edges
  - Draw the intermediate points of pairwise elements of the list.

Example

- Example:
  A: (30,10),B: (24,32),C: (20,22), D: (16,34)
  E: (8,26), F: (12,16)
- Define the polygon with

  A,B,C,D,E,F,A



Example

- Example:
  A: (30,10),B: (24,32),C: (20,22), D: (16,34)
  E: (8,26), F: (12,16)

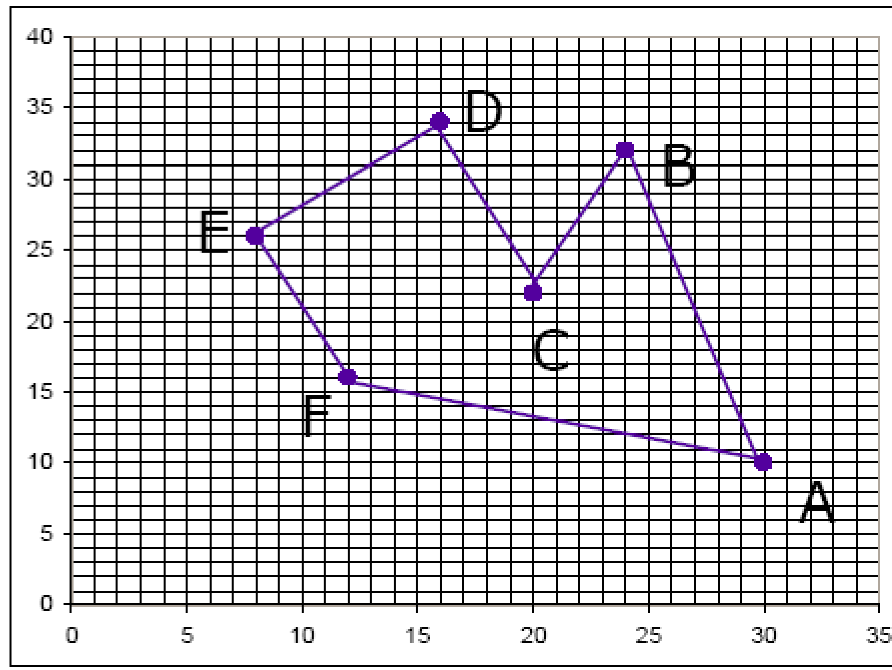
- Define the polygon with

  A,B,C,D,E,F,A

- E'=(8,25), F'=(12,15)

  Sorted Edge Table:

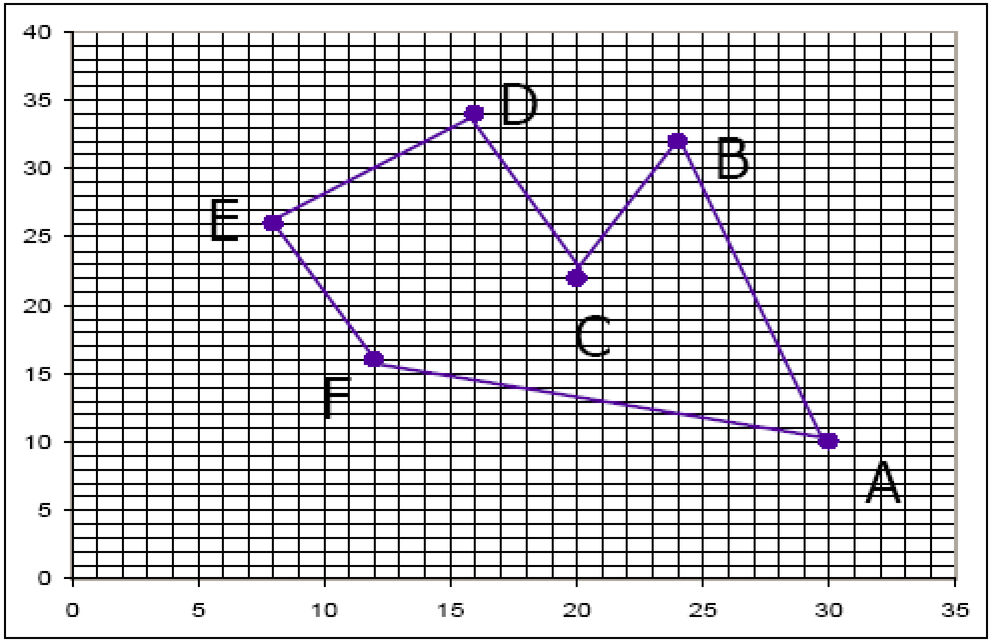
Y	E1	E2
10	[15,30,-3]	[32,30,-3/11]
16	[25,12,-2/5]	
22	[34,20,-1/3]	[32,20,2/5]
26	[34,8,1]	



Example

- Sorted Edge Table:

Y	E1	E2
10	[15,30,-3]	[32,30,-3/11]
16	[25,12,-2/5]	
22	[34,20,-1/3]	[32,20,2/5]
26	[34,8,1]	



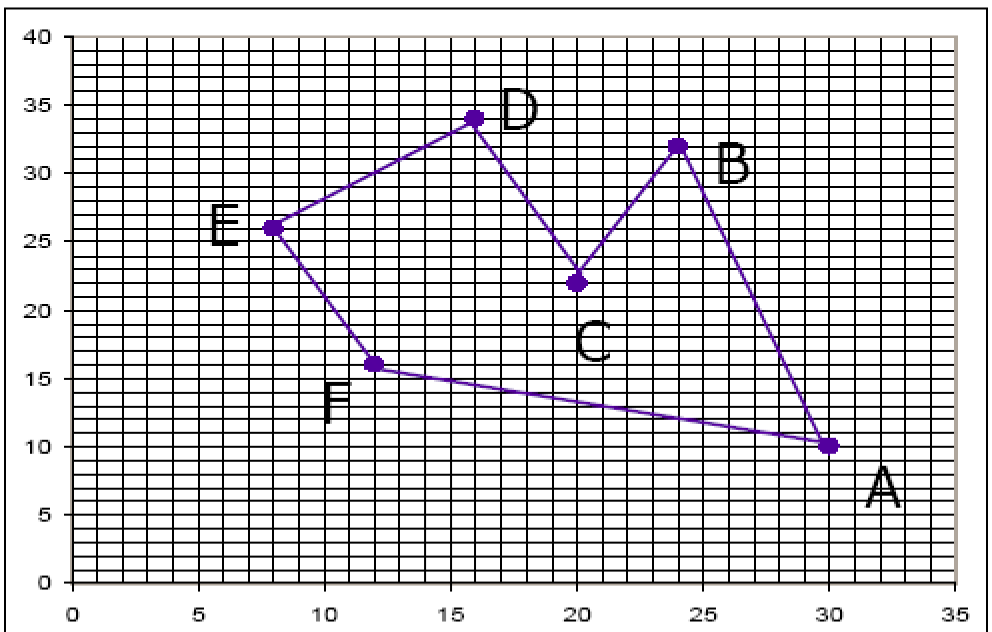
Active Edge List

Y	S1	S1	S2	S2
10	30	30		
11	27	29.73		
12	24	29.45		
13	21	29.18		
14	18	28.91		
15	15	28.64		
16	12	28.36		
17	11.6	28.09		
18	11.2	27.82		
19	10.8	27.55		
20	10.4	27.27		
21	10	27		
22	9.6	20	20	26.73
23	9.2	19.67	20.4	26.45
24	8.8	19.33	20.8	26.18
25	8.4	19	21.2	25.91
26	8	18.67	21.6	25.64
27	9	18.33	22	25.36
28	10	18	22.4	25.09
29	11	17.67	22.8	24.82
30	12	17.33	23.2	24.55
31	13	17	23.6	24.27
32	14	16.67	24	24
33	15	16.33		
34	16	16		

Example

- Sorted Edge Table:

Y	E1	E2
10	[15,30,-3]	[32,30,-3/11]
16	[25,12,-2/5]	
22	[34,20,-1/3]	[32,20,2/5]
26	[34,8,1]	



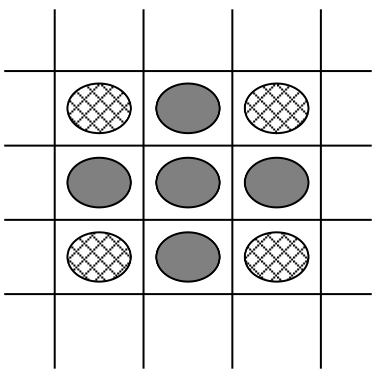
Active Edge List

Y	S1	S1	S2	S2
10	30	30		
11	27	29.73		
12	AF' 24	29.45		
13	21	29.18		
14	18	28.91		
15	15	28.64	AB	
16	12	28.36		
17	11.6	28.09		
18	11.2	27.82		
19	10.8	27.55		
20	FE' 10.4	27.27		
21	10	27		
22	9.6	20	20	26.73
23	9.2	19.67	20.4	26.45
24	8.8	19.33	20.8	26.18
25	8.4	CD 19	21.2	25.91
26	8	18.67	21.6	25.64 AB
27	9	18.33	CB 22	25.36
28	10	18	22.4	25.09
29	ED 11	17.67	22.8	24.82
30	12	17.33	23.2	24.55
31	13	17	23.6	24.27
32	14	16.67	24	24
33	15	16.33		
34	16	16		

Area Filling – for irregular areas

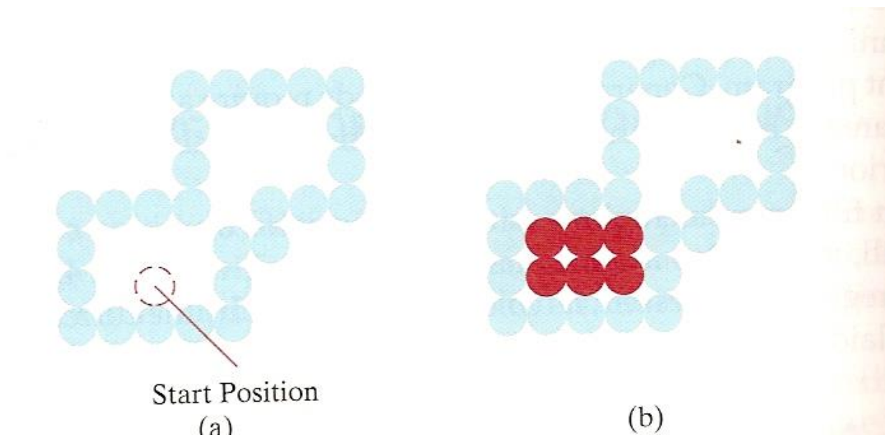
- Paint the interior point by point out to the boundary.

- Pixel Adjacency



4-connected

8-connected

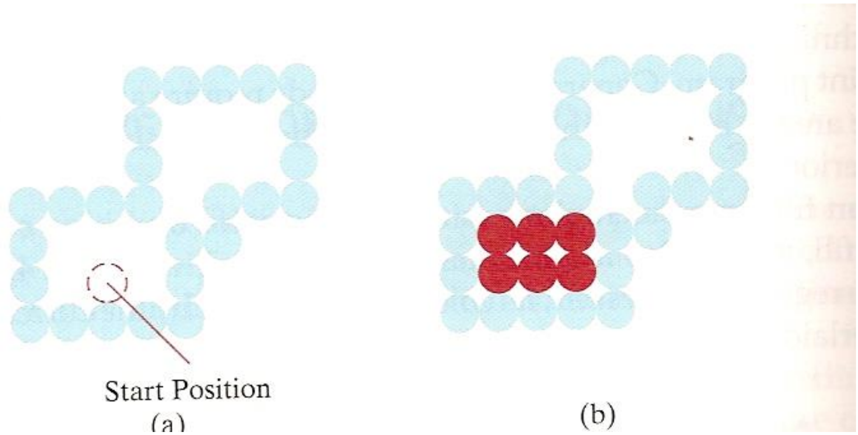


Area Filling – for irregular areas

- Boundary-Fill Algorithm
  - starting at a point inside the figure and painting the interior in a specified color or intensity.

Used in interactive painting packages

Like identifying connected components



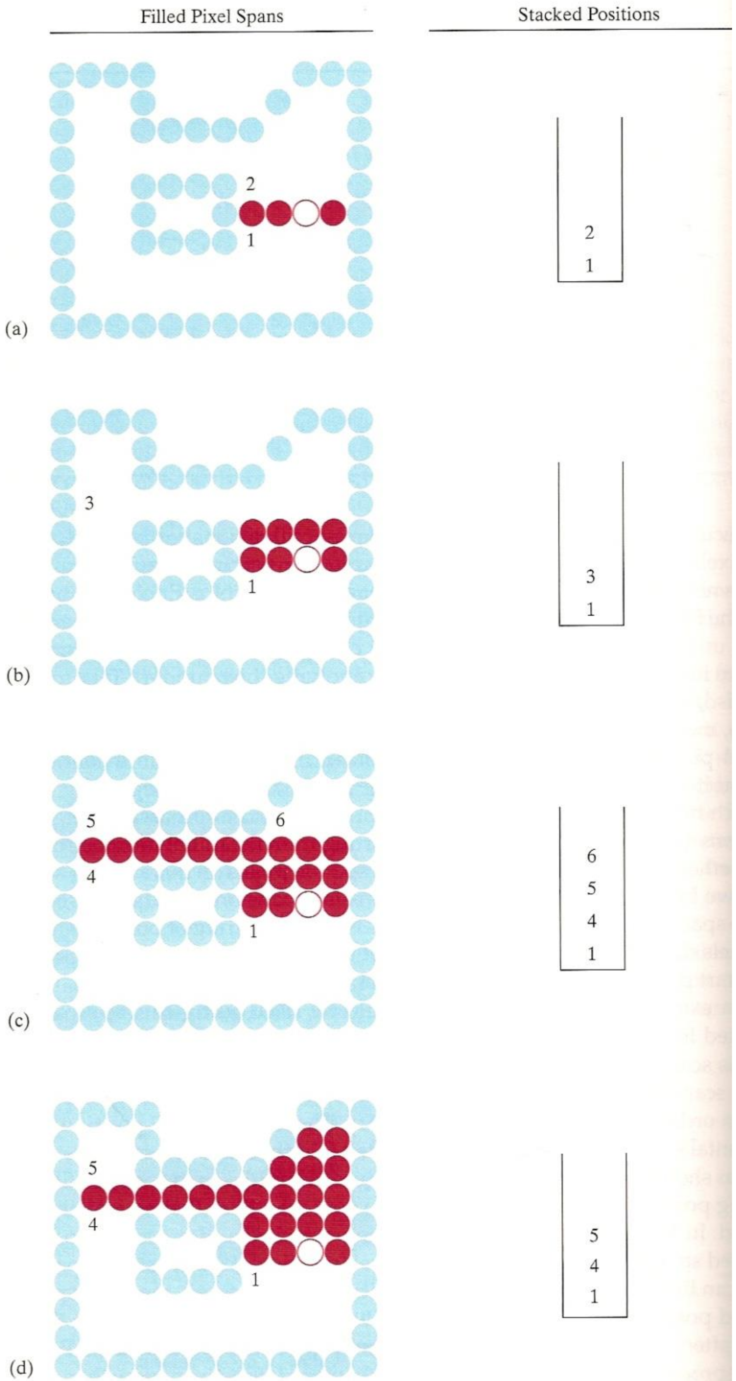
Boundary Fill Algorithm

- Start at a point inside a continuous arbitrary shaped region and paint the interior outward toward the boundary. Assumption: boundary color is a single color
- (x,y): start point; b:boundary color, fill: fill color

```
void boundaryFill4(x,y,fill,b) {
  cur = getpixel(x,y)
  if (cur != b) AND (cur != fill) {
        setpixel(x,y,fill);
        boundaryFill4(x+1,y,fill,b);
        boundaryFill4(x-1,y,fill,b);
        boundaryFill4(x,y+1,fill,b);
        boundaryFill4(x,y-1,fill,b);
  }
}
```

- 4 neighbors vs 8 neighbors: depends on definition of continuity.
  8 neighbor: diagonal boundaries will not stop the fill
- Recursive, so slow. For large regions with millions of pixels, millions of function calls.
- Stack based improvement: keep neighbors in stack
- Number of elements in the stack can be reduced by filling the area as pixel spans and pushing only the pixels with pixel transitions.

Boundary Filling



Flood-Fill

- Similar to boundary fill. Can be used for cases when the boundary is not single-color. Algorithm continues while the neighbor pixels have the same color.

-
```
void FloodFill4(x,y,fill,oldcolor) {
  cur = getpixel(x,y)
  if (cur == oldcolor) {
        setpixel(x,y,fill);
        FloodFill4(x+1,y,fill,oldcolor);
        FloodFill4(x-1,y,fill,oldcolor);
        FloodFill4(x,y+1,fill,oldcolor);
        FloodFill4(x,y-1,fill,oldcolor);
  }
}
```

Fill pattern

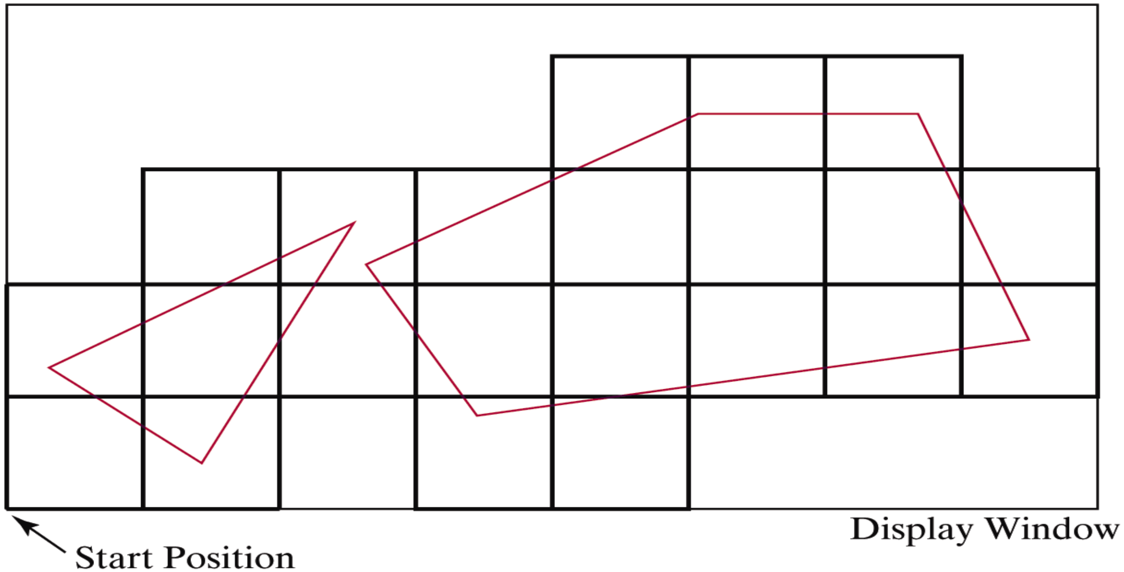


Figure 4-31

Tiling a rectangular fill pattern across a display window to fill two convex polygons.

Fill pattern

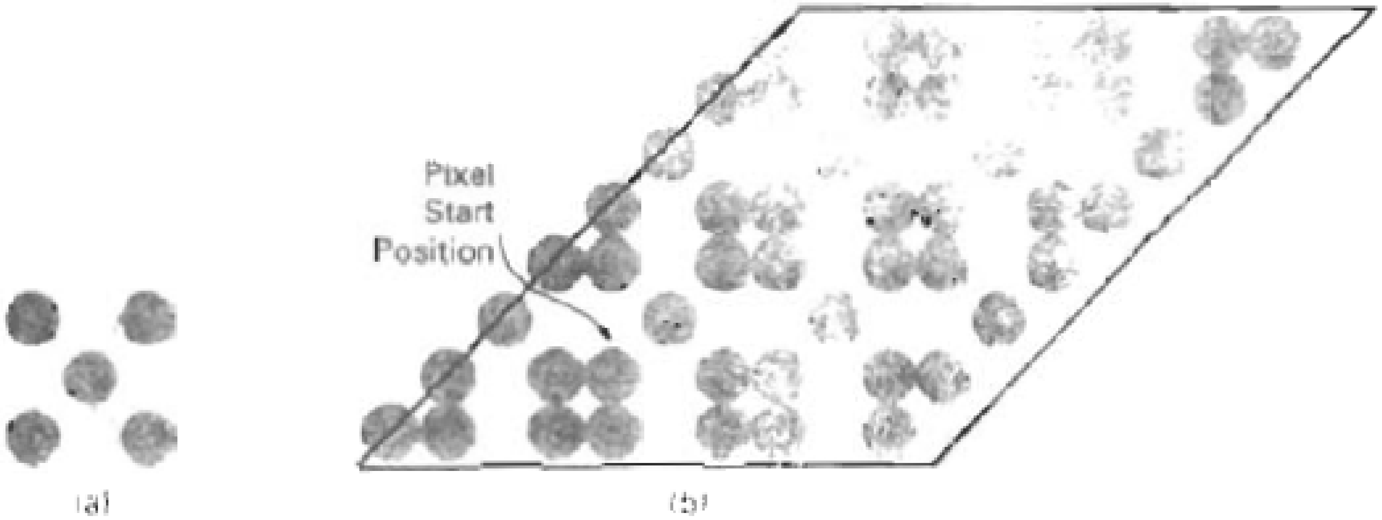


Figure 4-22
A pattern array (a) superimposed on a parallelogram fill area to produce the display (b).

Hollow display of a polygon

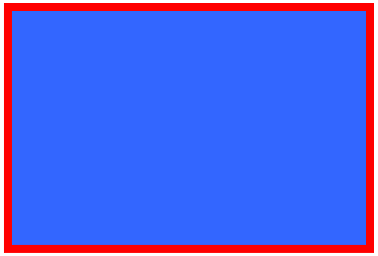
glPolygonMode (face, displayMode);

Face: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK

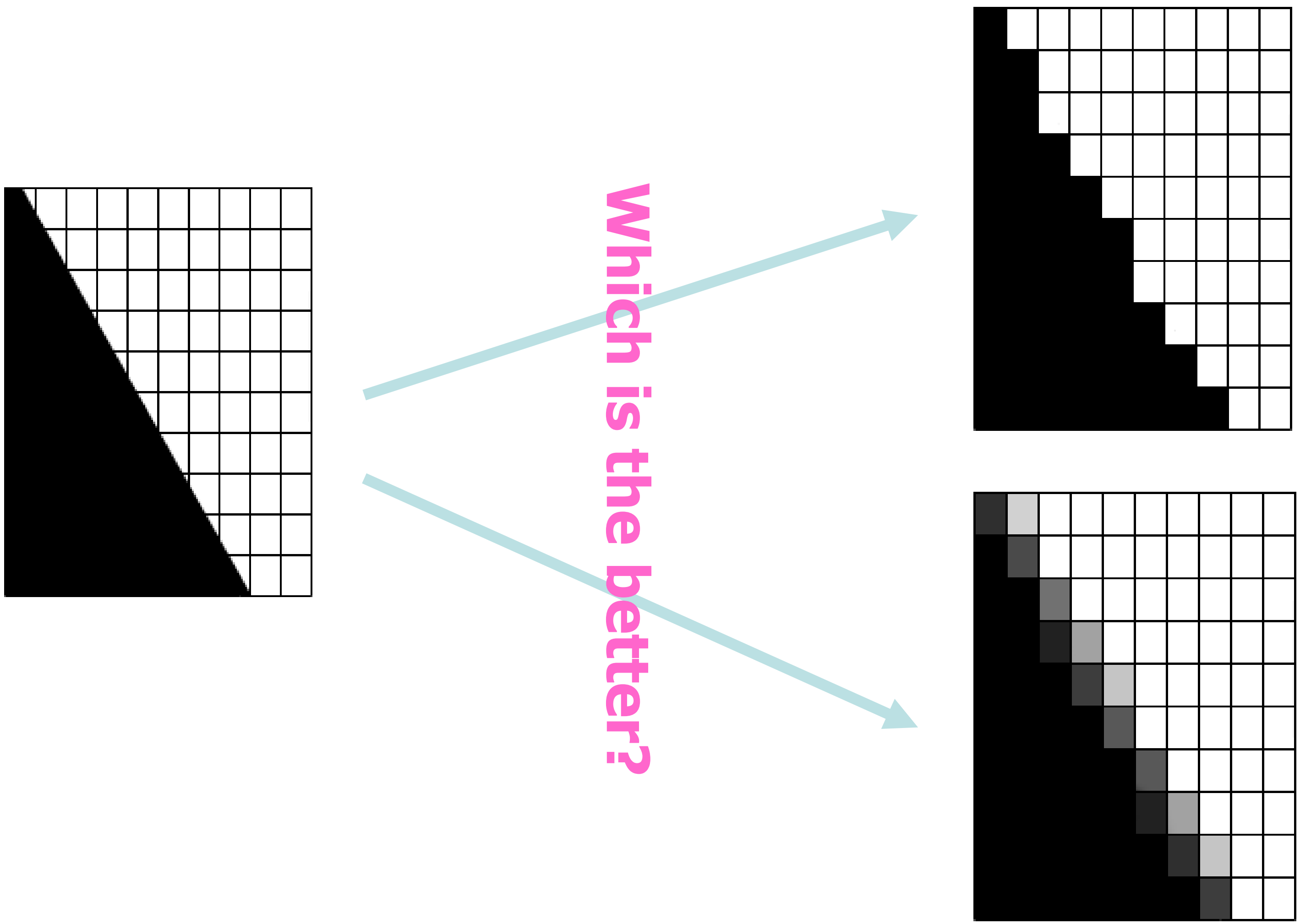
displayMode :	GL_LINE	no fill
	GL_POINTS	vertex points only
	GL_FILL	fill (default)

```
glColor3f (0.0, 1.0, 0.0);    \\blue fill
\\generate polygon

glColor3f (1.0, 0.0, 0.0);    \\red line
glPolygonMode (GL_FRONT, GL_LINE);
 \\generate polygon
```



Aliasing in CG

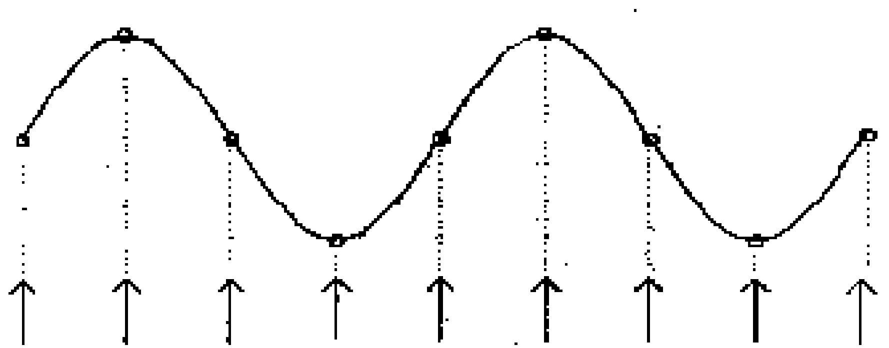


Aliasing in Computer Graphics

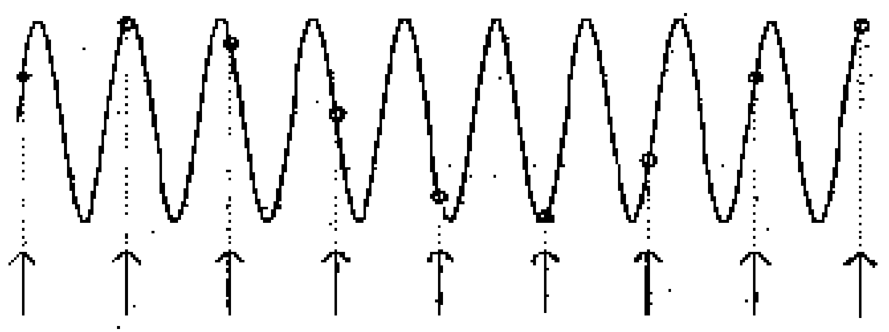
- Digital technology can only *approximate* analogue signals through a process known as *sampling*
- The distortion of information due to low-frequency sampling (undersampling)
- Choosing an appropriate *sampling rate* depends on data size restraints, need for accuracy, the cost per sample…
- Errors caused by aliasing are called **artifacts**. Common aliasing artifacts in computer graphics include jagged profiles, disappearing or improperly rendered fine detail, and disintegrating textures.

The Nyquist Theorem

The sampling rate must be at least twice the frequency of the signal or *aliasing* occurs (twice the frequency of the highest frequency component)



(a) Point sampling within the Nyquist limit



(b) Point sampling beyond the Nyquist limit

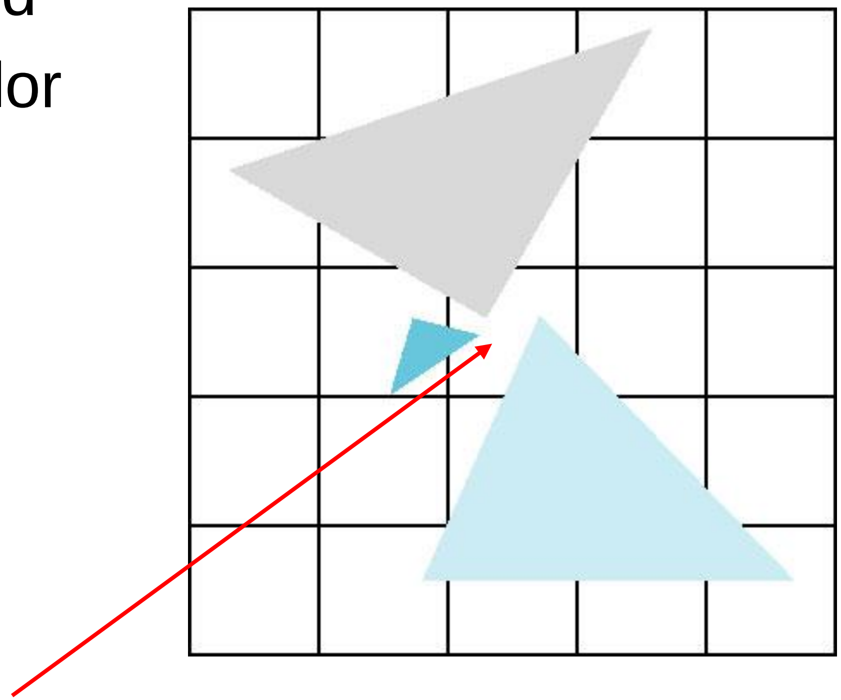
Aliasing

- In order to have any hope of accurately reconstructing a function from a periodically sampled version of it, two conditions must be satisfied:
  - The function must be bandlimited.
  - The sampling frequency must be at least twice the maximum frequency of the function.

Satisfying these conditions will eliminate aliasing.

Polygon Aliasing

- Aliasing problems can be serious for polygons
  - Jaggedness of edges
  - Small polygons neglected
  - Need compositing so color of one polygon does not totally determine color of pixel



All three polygons should contribute to color

Antialiasing

- Antialiasing methods try to combat the effects of aliasing.
- Two major categories of antialiasing techniques
  - prefiltering
  - postfiltering.

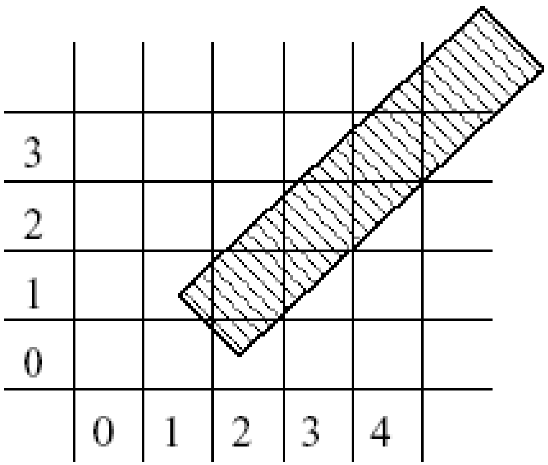
Prefiltering

- Central idea is to consider the pixel as an area, not a point, and calculate the pixel colour value from a weighted sum of the colours present inside the area.
  - e.g. In line-drawing, think of the line as a long rectangle, not the minimal line of pixels found by Bresenham’s method.Pixels partly covered by this rectangle would be shaded grey

Prefiltering (cont'd)

- In the sample figure, the percentage of the pixel area covered by the line segment is used to determine the intensity level of the pixel.

- If multiple objects contribute to one pixel, then the intensities are weighed appropriately.



Pixel	% of Full Intensity
(1,0)	10%
(1,1)	20%
(2,0)	20%
(2,1)	90%
(2,2)	40%
(3,1)	50%
(3,2)	90%
(3,3)	30%

Anti-aliasing through area averaging

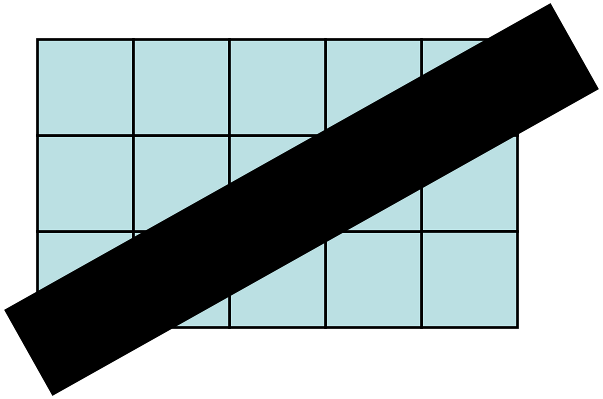
How can we make the line less jagged and avoid aliasing?

Bresenham's algorithm is optimal for drawing the line if you only have 2 colors.   It chooses the closest set of pixels to the line.

However, if you have more than 2 colors, you can color the pixels differently depending on the distance to the line.

Suppose each pixel is square, 1 unit high and 1 unit wide.
Suppose the line is 1 unit in width.

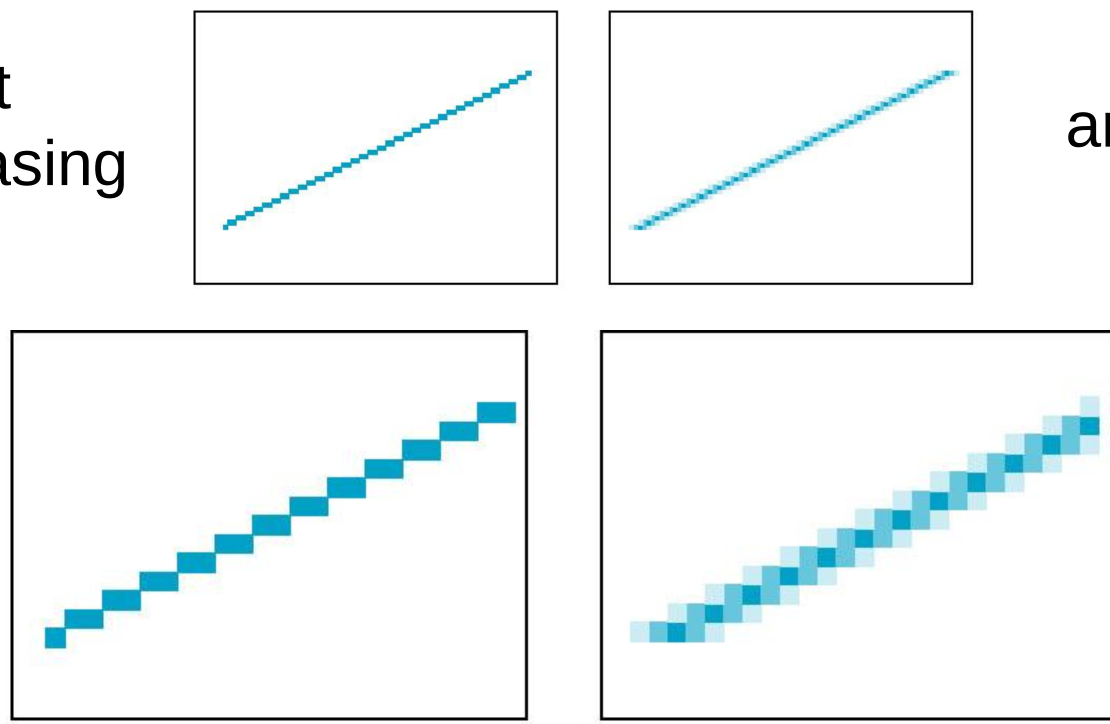
Shade each pixel according to the percentage of that pixel covered by the line.



Antialiasing by Area Averaging

- Color multiple pixels for each x depending on coverage by ideal line

without antialiasing



antialiased

magnified

Postfiltering (Supersampling)

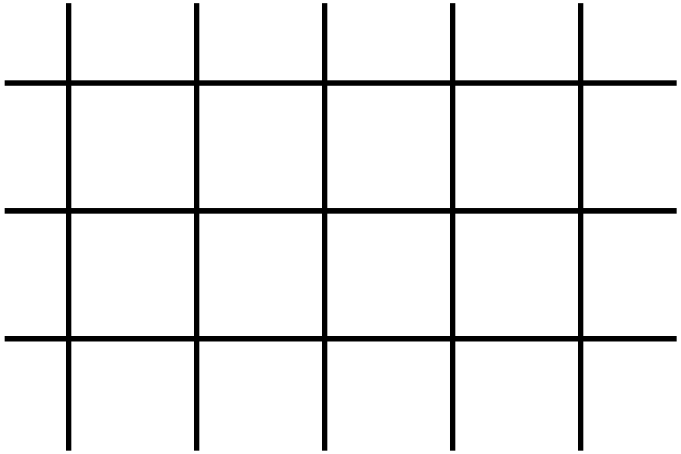
- Postfiltering is also known as “point sampling”.
- In supersampling, more than one sample per pixel is evaluated.
- For example, one might compute a $2K \times 2K$ or $4K \times 4K$ image and display it on a $1K \times 1K$ monitor.
- Note that $n \times n$ supersampling increases the number of samples and the image-generation time by a factor of n^2!
- Supersampling has the effect of moving the Nyquist Limit to higher frequencies.
- Note that even if we move the Nyquist Limit to higher frequencies, we will still have aliasing.

Antialiasing (postfiltering)

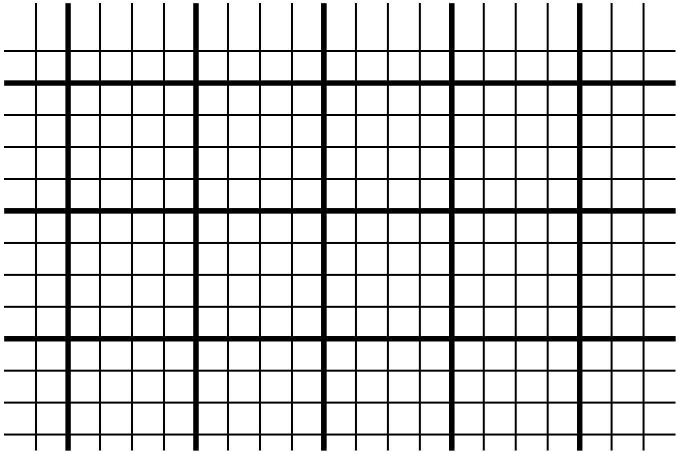
- Postfiltering first supersamples the signal in its unfiltered form and then filters out the high frequency from the supersamples.
- Increase the sampling rate by treating the screen as if it had a finer grid resolution than is actually available

(non-adaptive) Super-Sampling

- Split single pixel into sub-pixels.
- Pixel's final color is a mixture of sub-pixels' colors. Simple method: Sample at the middle of each sub-pixel. Then, pixel's color is the average of the sub-pixels' color.

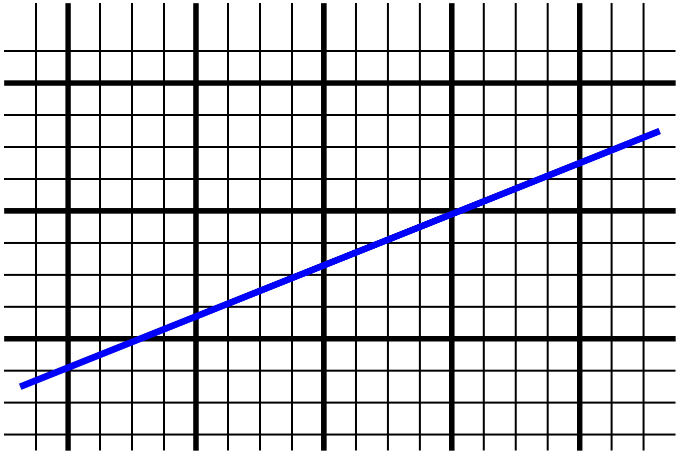


Pixels



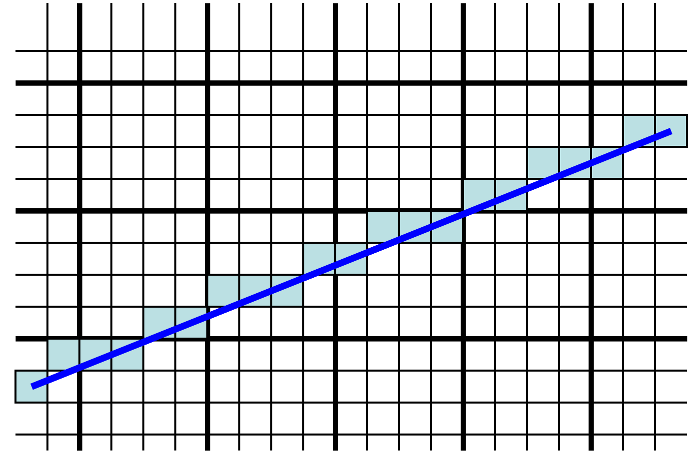
Sub-divide into sub-pixels

Super-Sampling a Zero-Width Line



Sub-divide each pixel into sub-pixels, for example 4x4 sub-pixels

Apply Bresenham's algorithm at sub-pixel level



Each pixel can have a maximum of 4 colored sub-pixels

0	0	0	4
2	4	4	0
2	0	0	0

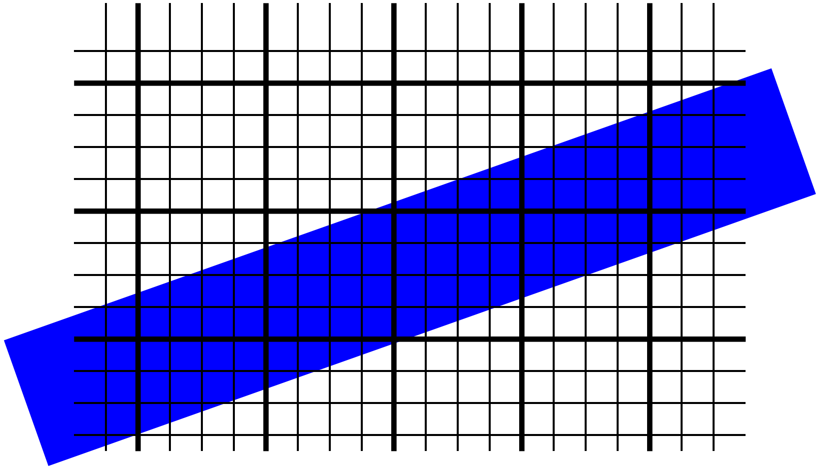
How many sub-pixels are colored?

Assign color

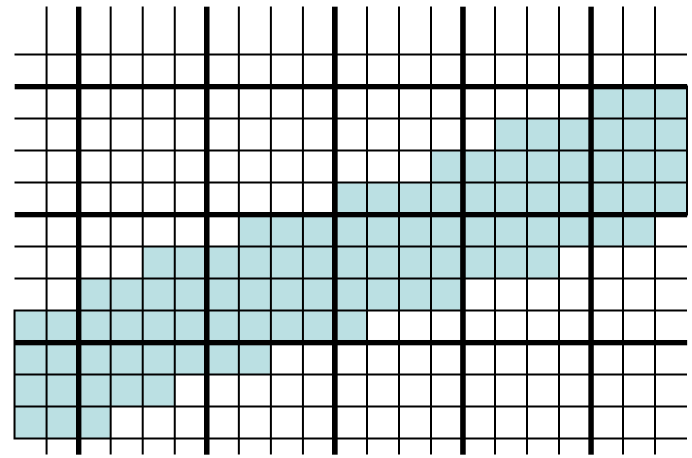
0.0	0.0	0.0	1.0
0.5	1.0	1.0	0.0
0.5	0.0	0.0	0.0

Fraction of pixel's color to be line's color

Super-Sampling a Line with Non-Zero Width



A line that is one-pixel wide.
For every pixel: Maximum number of sub-pixels inside line = 16



A sub-pixel is considered in if its lower-left corner is inside the line

0	0	5/16	9/16
10/16	15/16	13/16	7/16
8/16	2/16	0	0

Fraction of sub-pixels are in = fraction of color of the pixel should be line color