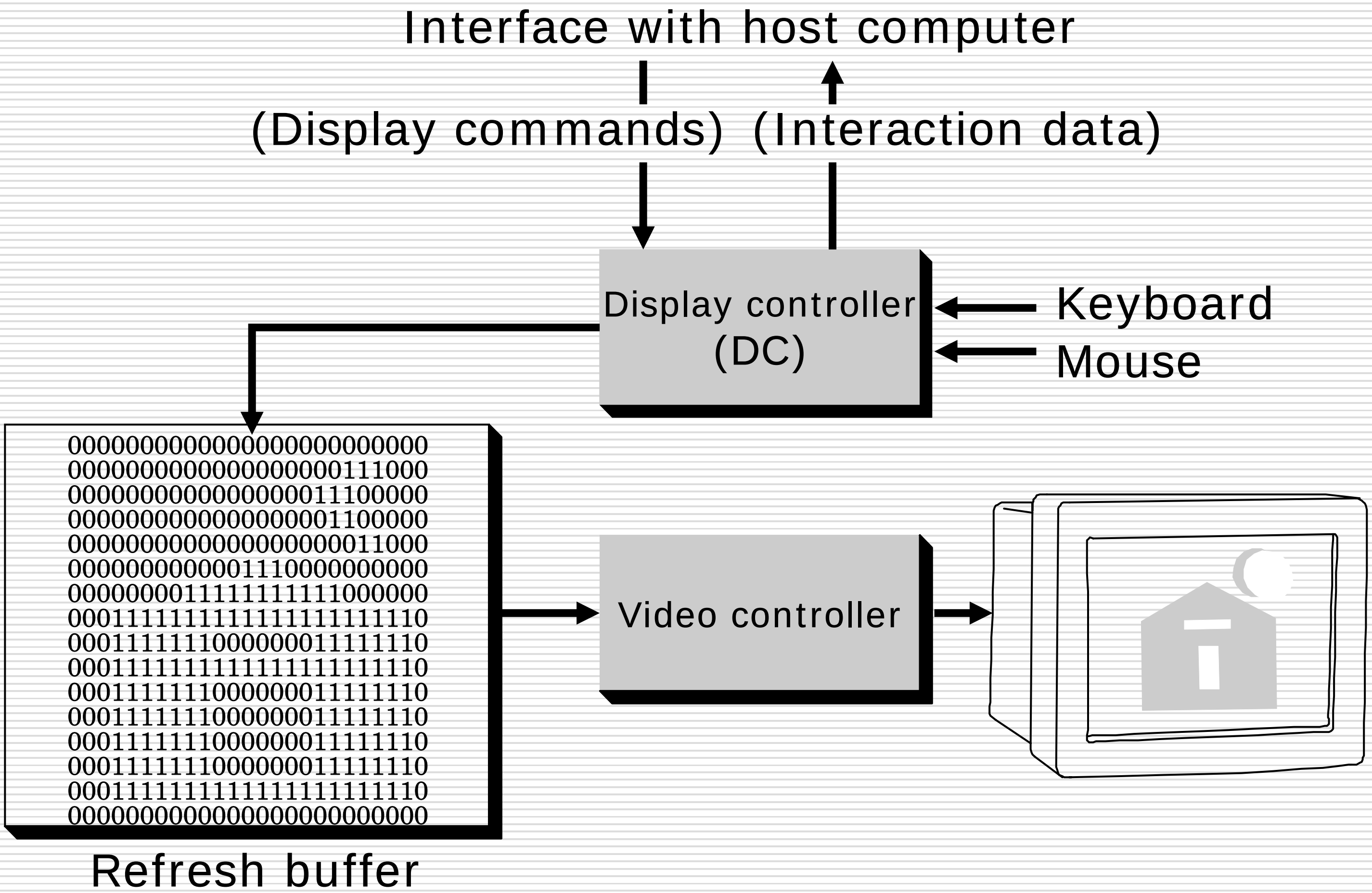
Computer Graphics

Jeng-Sheng Yeh 葉正聖
Ming Chuan University
(modified from Bing-Yu Chen's slides)

Basic Raster Graphics Algorithms for Drawing 2D Primitives

- Architecture of a Raster Display
- Scan Converting Lines
- Filling Rectangles
- Filling Polygons
- Clipping Lines
- Clipping Polygons
- Antialiasing

Architecture of a Raster Display



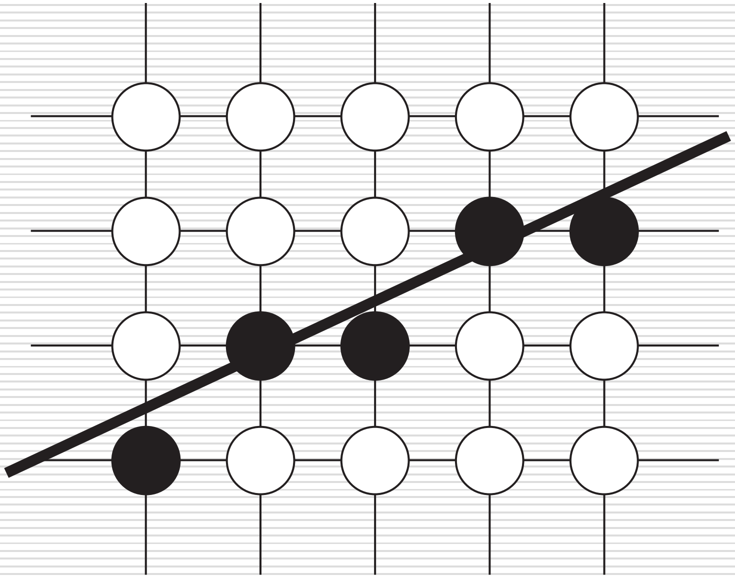
Definitions

- Pixel
  - a screen consists of N x M pixels
- Bilevel
  - = monochrome, 1 bit / pixel
- Color: RGB model
  - 16bits / pixel
    - R, G, B each 5 bits, 1 bit overlay
  - 24bits / pixel
    - R, G, B each 8 bits
  - 8 bits / pixel
    - 256 colors, color map, indexing

Definitions

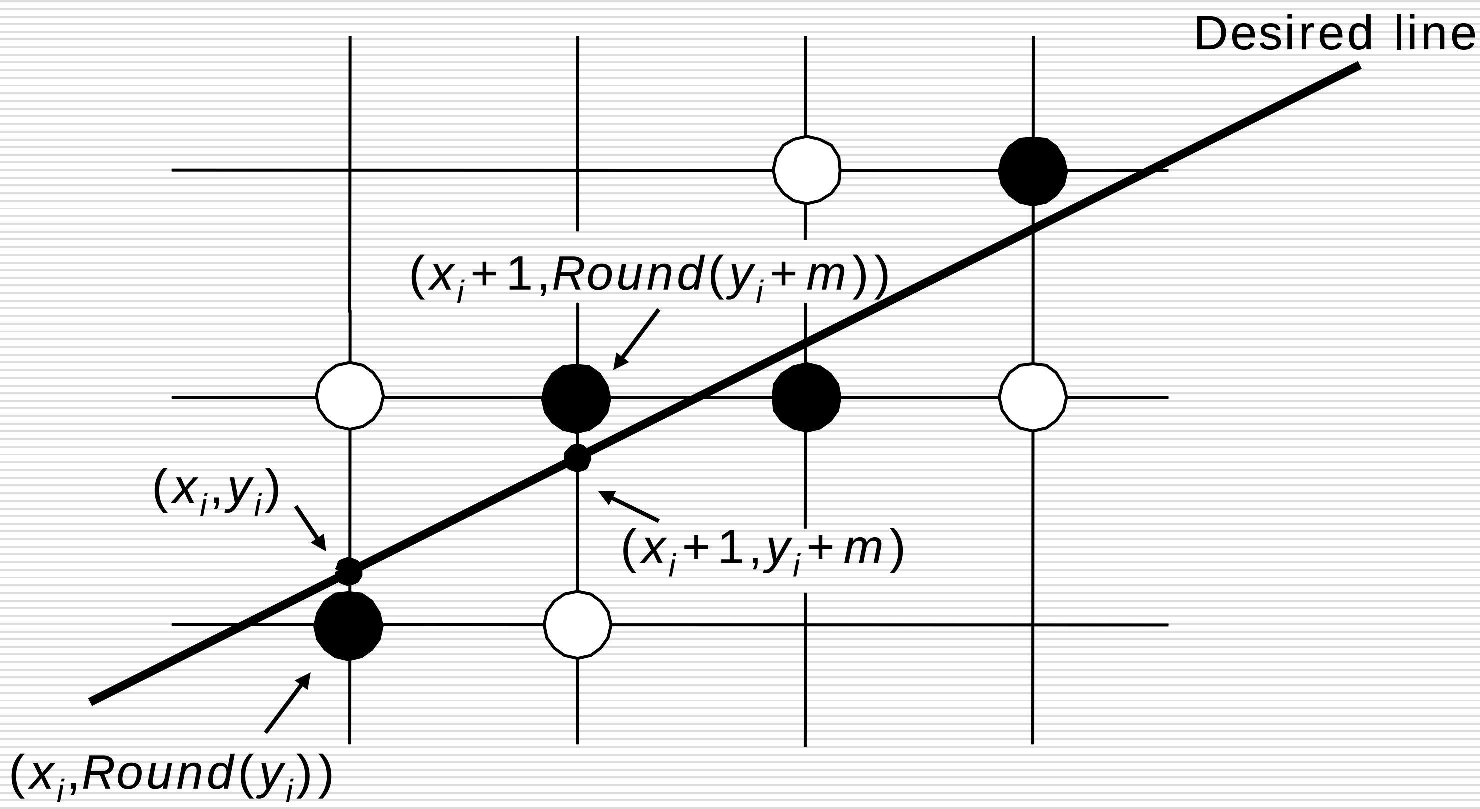
- bitmap / pixmap
  - bitmap
    - 1-bit-per-pixel bilevel systems
  - pixmap
    - multiple-bit-per-pixel systems
- frame buffer
  - an array of data in memory mapped to screen

Scan Converting Lines



- A scan-converted line showing intensified pixels as black circles

The Basic Incremental Algorithm



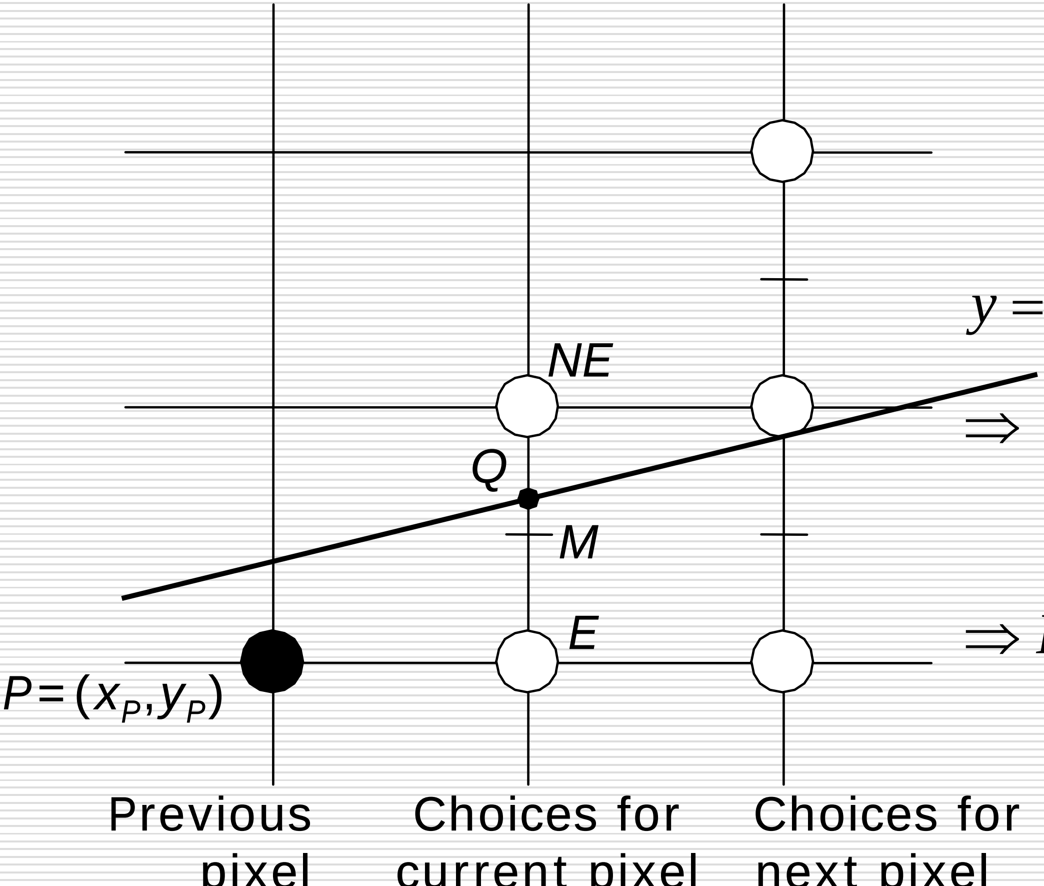
$$y_{i+1} = mx_{i+1} + B = m(x_i + \Delta x) + B = y_i + m\Delta x$$

The Basic Incremental Algorithm

```
void Line (int x0, int y0, int x1, int y1, value) {
    int x;
    float dy, dx, y, m;

    dy=y1-y0;
    dx=x1-x0;
    m=dy/dx;
    y=y0;
    for (x=x0; x<=x1; x++) {
        WritePixel (x, (int)floor(y+0.5), value);
        y+=m;
    }
}
```

Midpoint Line Algorithm



$$y = \frac{dy}{dx}x + B$$

$$\Rightarrow \quad F(x, y) = dy \cdot x - dx \cdot y + B \cdot dx = 0$$

$$= a \cdot x + b \cdot y + c$$

$$\Rightarrow F(M) = F(x_P + 1, y_P + \frac{1}{2}) = d$$

Midpoint Line Algorithm

$$d_{old} = F(x_P + 1, y_P + \frac{1}{2}) = a(x_P + 1) + b(y_P + \frac{1}{2}) + c$$

$$d_{new} = \begin{cases} F(x_P + 2, y_P + \frac{1}{2}) = a(x_P + 2) + b(y_P + \frac{1}{2}) + c & \text{for E} \\ F(x_P + 2, y_P + \frac{3}{2}) = a(x_P + 2) + b(y_P + \frac{3}{2}) + c & \text{for NE} \end{cases}$$

$$d_{new} = \begin{cases} d_{old} + a & \text{for E} \\ d_{old} + a + b & \text{for NE} \end{cases}$$

$$d_0 = F(x_0 + 1, y_0 + \frac{1}{2}) = a + \frac{b}{2} = dy - \frac{dx}{2}$$

Midpoint Line Algorithm

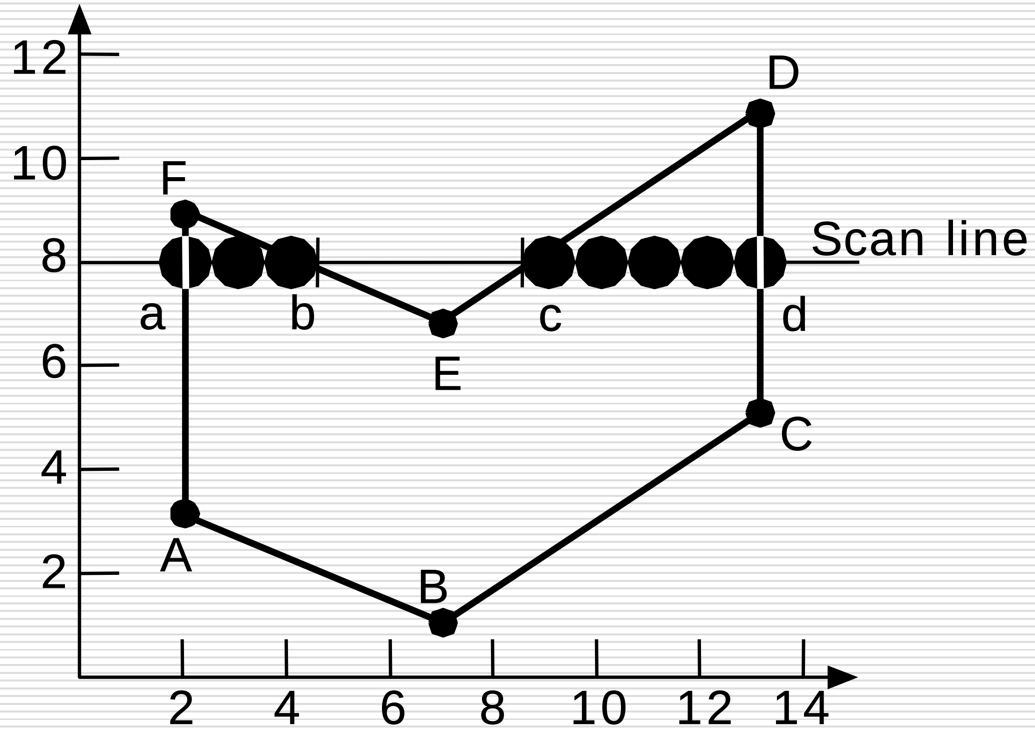
```
void MidpointLine (int x0, int y0, int x1, int y1, value) {
    int dx, dy, incrE, incrNE, d, x, y;

    dy=y1-y0;           dx=x1-x0;                d=dy*2-dx;
    incrE=dy*2;         incrNE=(dy-dx)*2;
    x=x0;               y=y0;
    WritePixel (x, y, value);
    while (x<x1) {
        if (d<=0) {     d+=incrE;      x++;
        } else {        d+=incrNE;     x++;    y++;
        }
        WritePixel (x, y, value);
    }
}
```

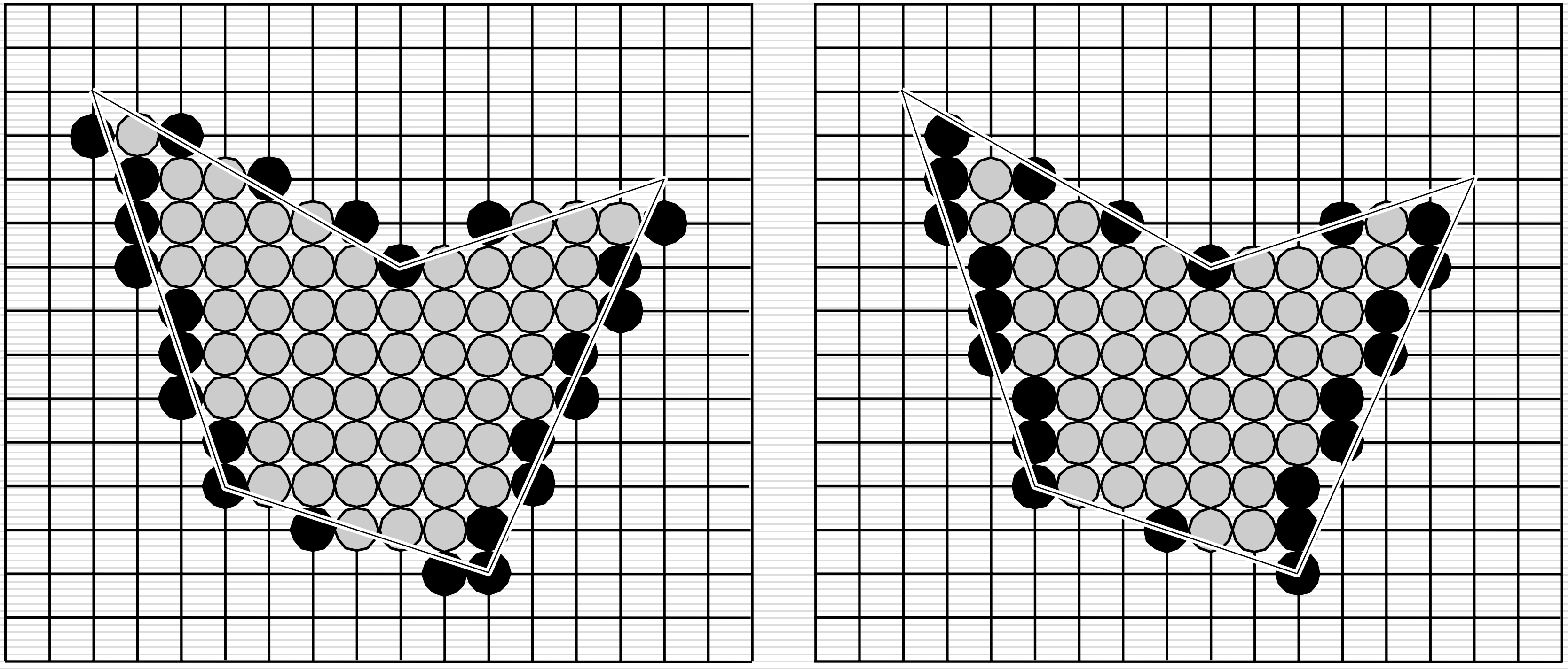
Filling Rectangles

```
for (y from ymin to ymax of the rectangle) {
    for (x from xmin to xmax) {
        WritePixel (x, y, value);
    }
}
```

Filling Polygons



Filling Polygons



Span extrema   Other pixels in the span

Filling Polygons

1. find the intersections of the scan line with all edges of the polygon
2. sort the intersections by increasing *x* coordinate
3. fill in all pixels between pairs of intersections that lie interior to the polygon

Step 3 requires 4 elaborations

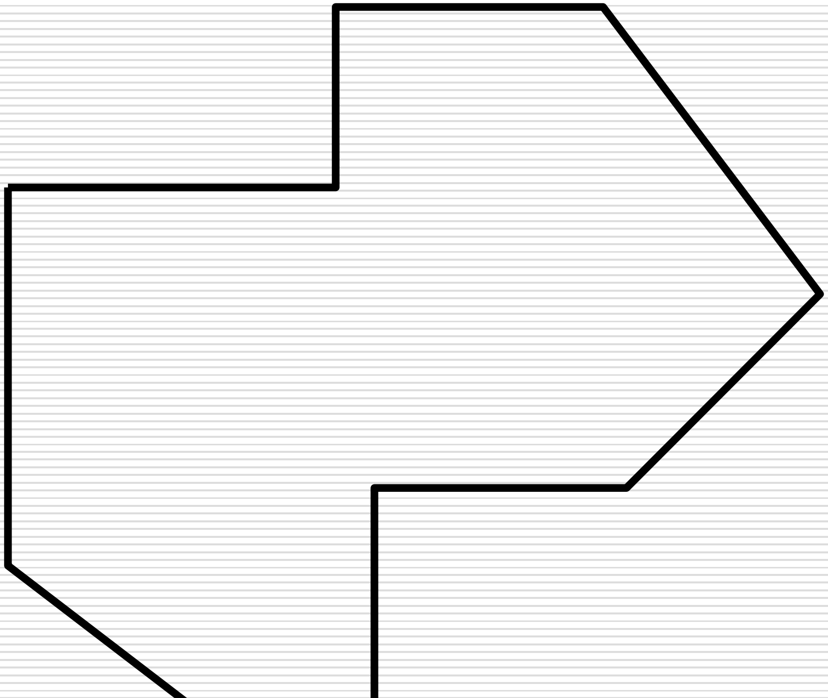
3.1 given an intersection with an arbitrary, fractional x value, how do we determine which pixel on either side of that intersection is interior ?

3.2 how do we deal with the special case of intersections at integer pixel coordinates ?

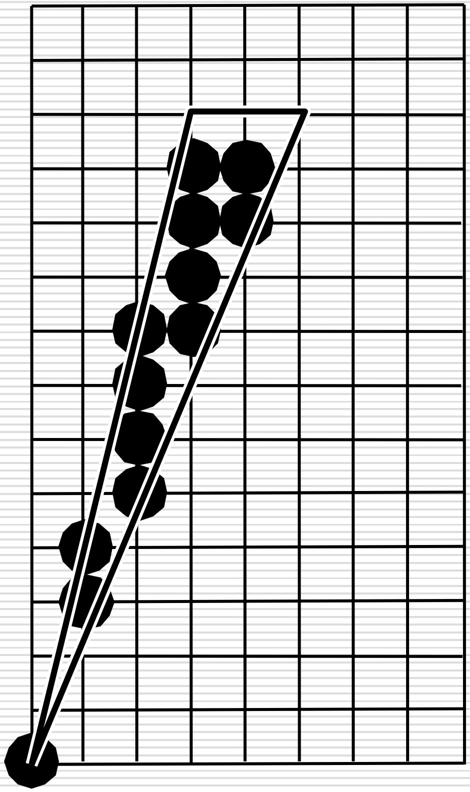
3.3 how do we deal with the special case in step 3.2 for shared vertices ?

3.4 how do we deal with the special case in step 3.2 in which the vertices define a horizontal edge ?

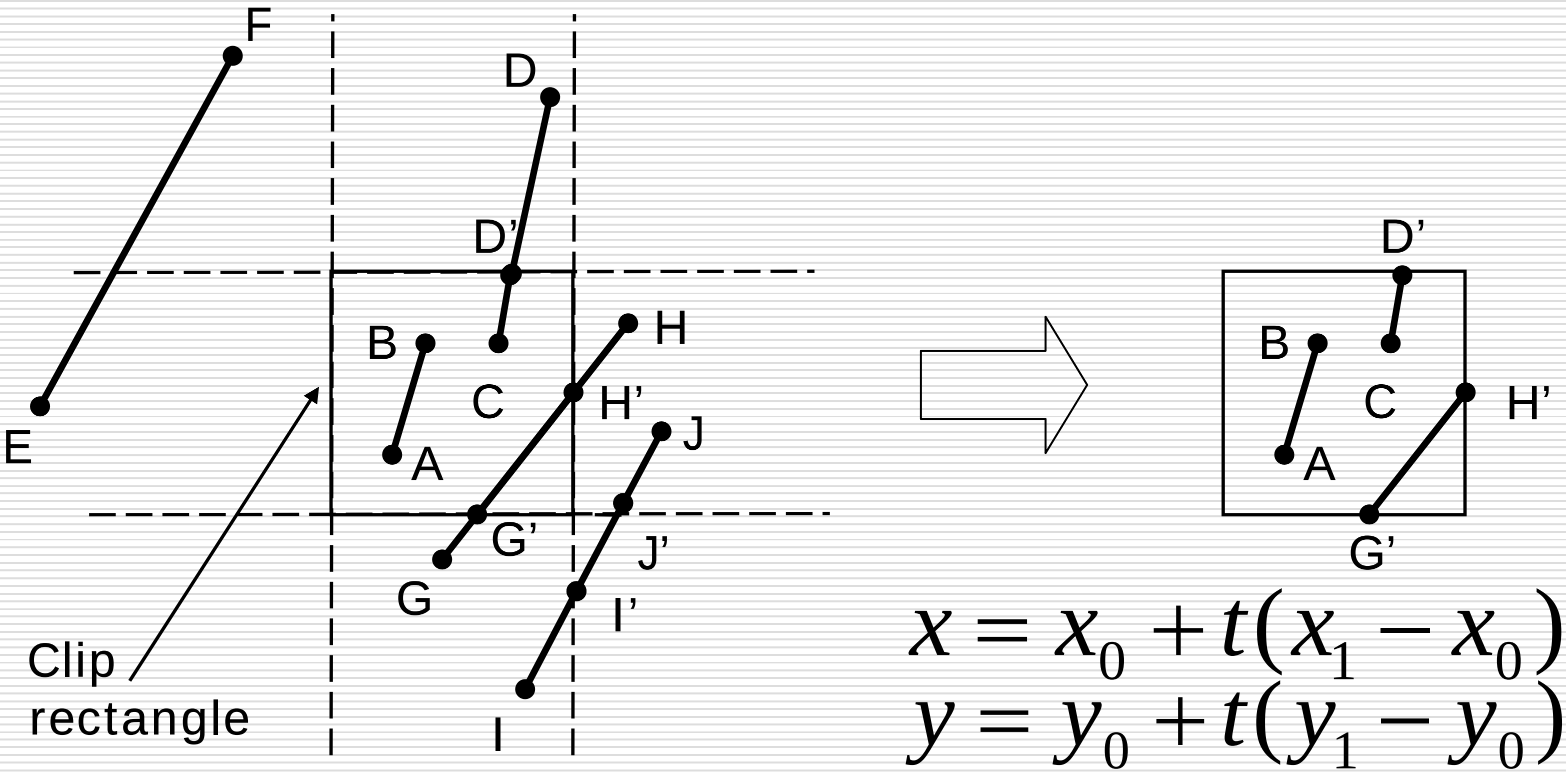
Horizontal Edges



Slivers

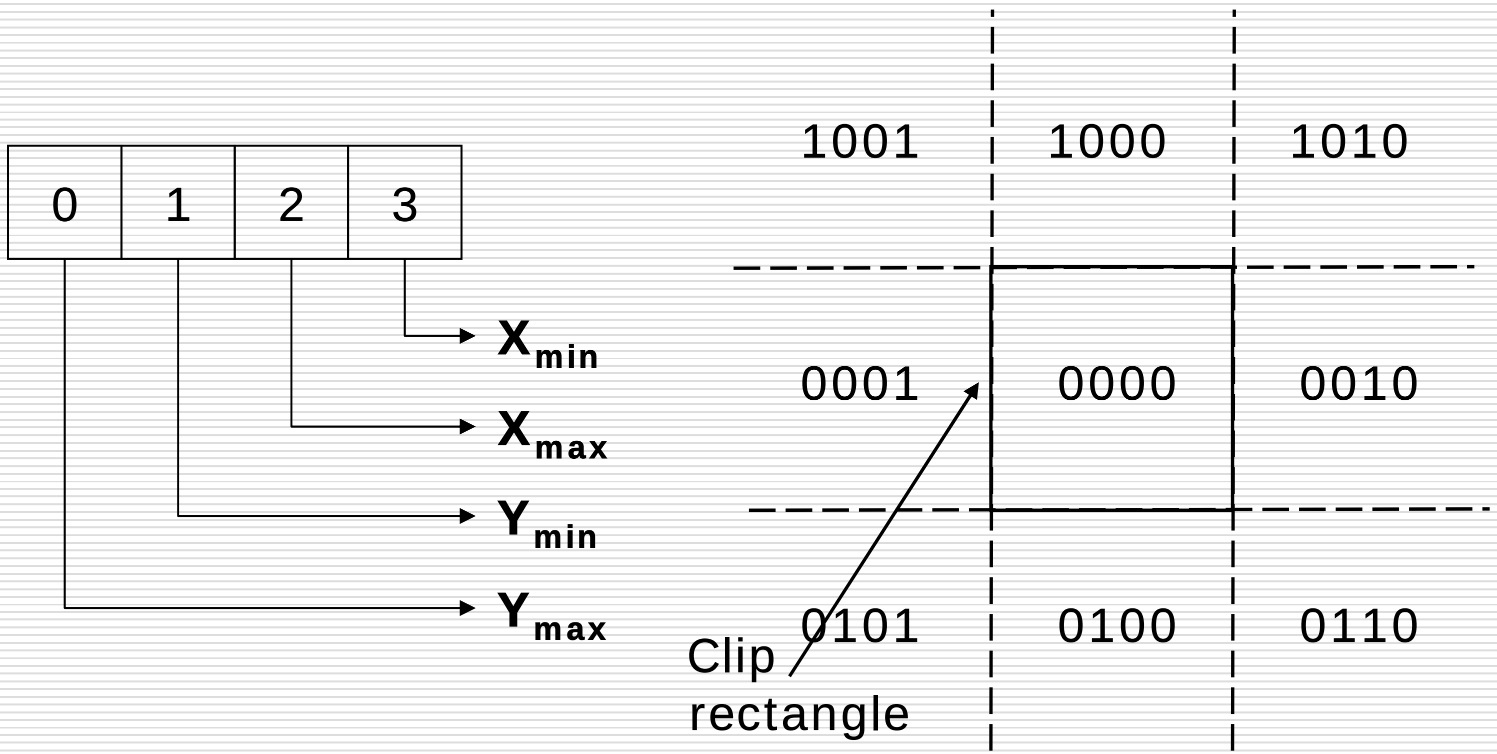


Clipping Lines

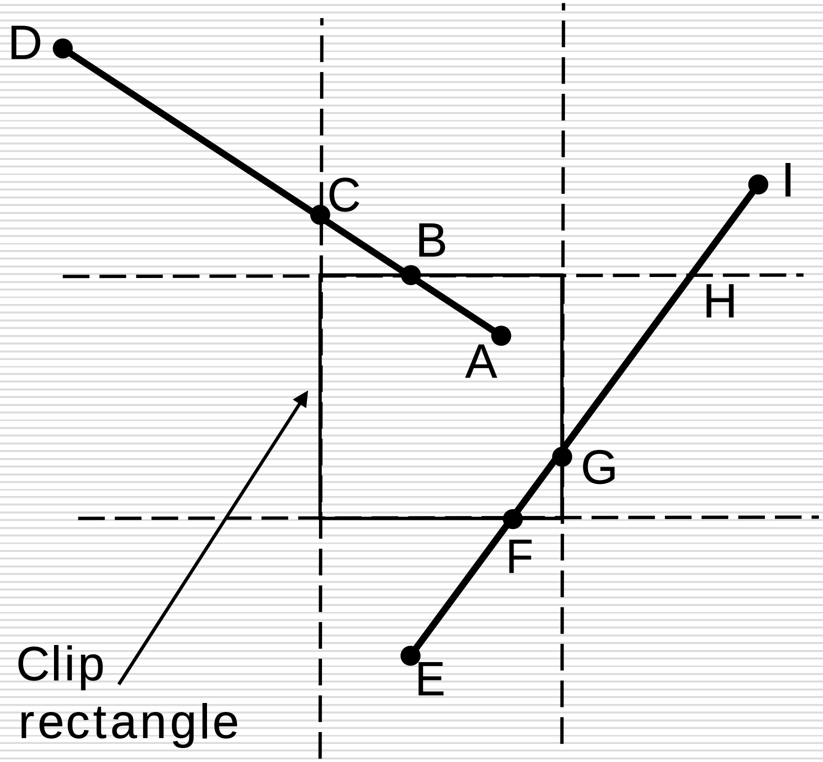


$$x = x_0 + t(x_1 - x_0)$$
$$y = y_0 + t(y_1 - y_0)$$

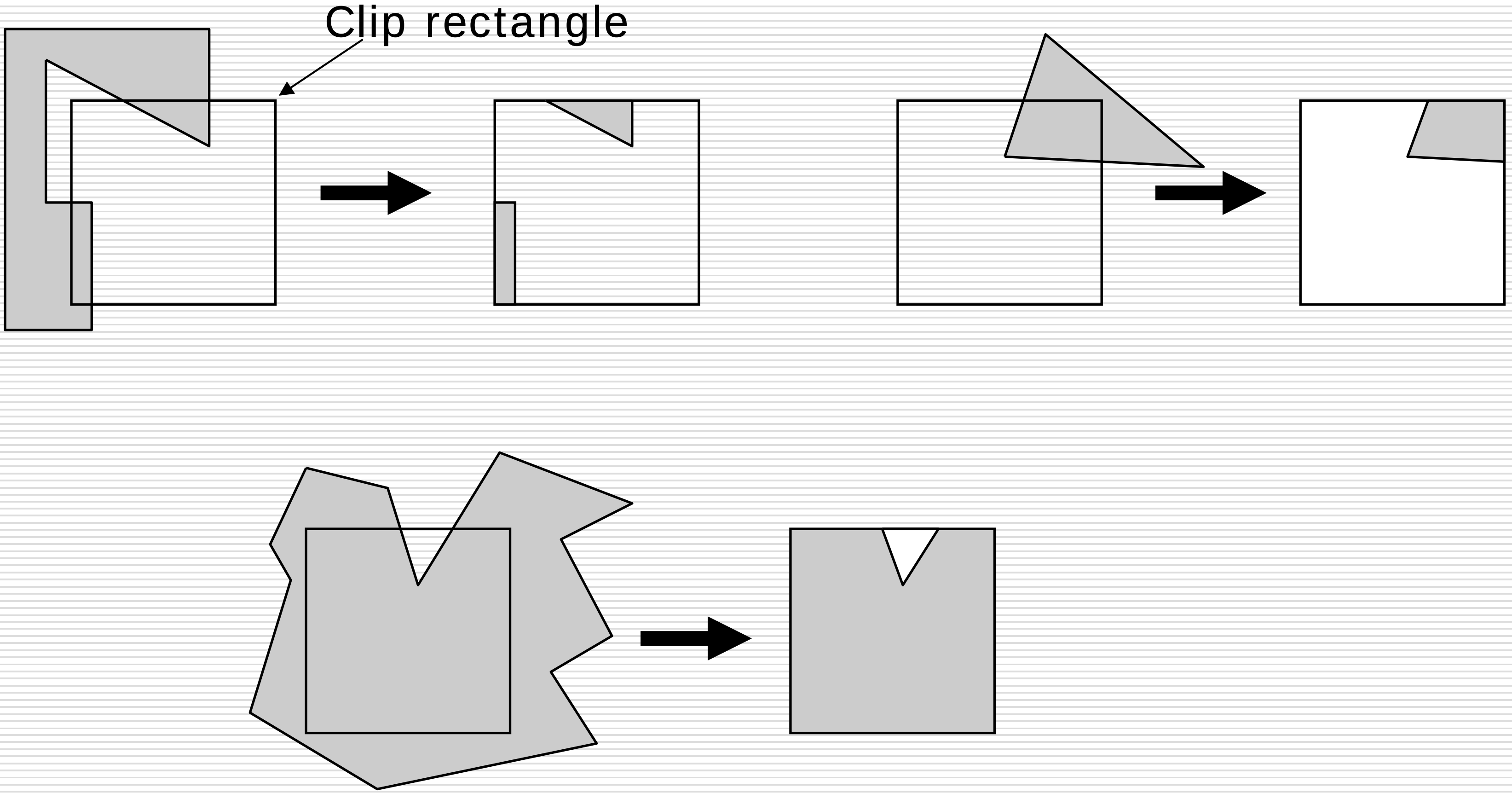
The Cohen-Sutherland Line-Clipping Algorithm



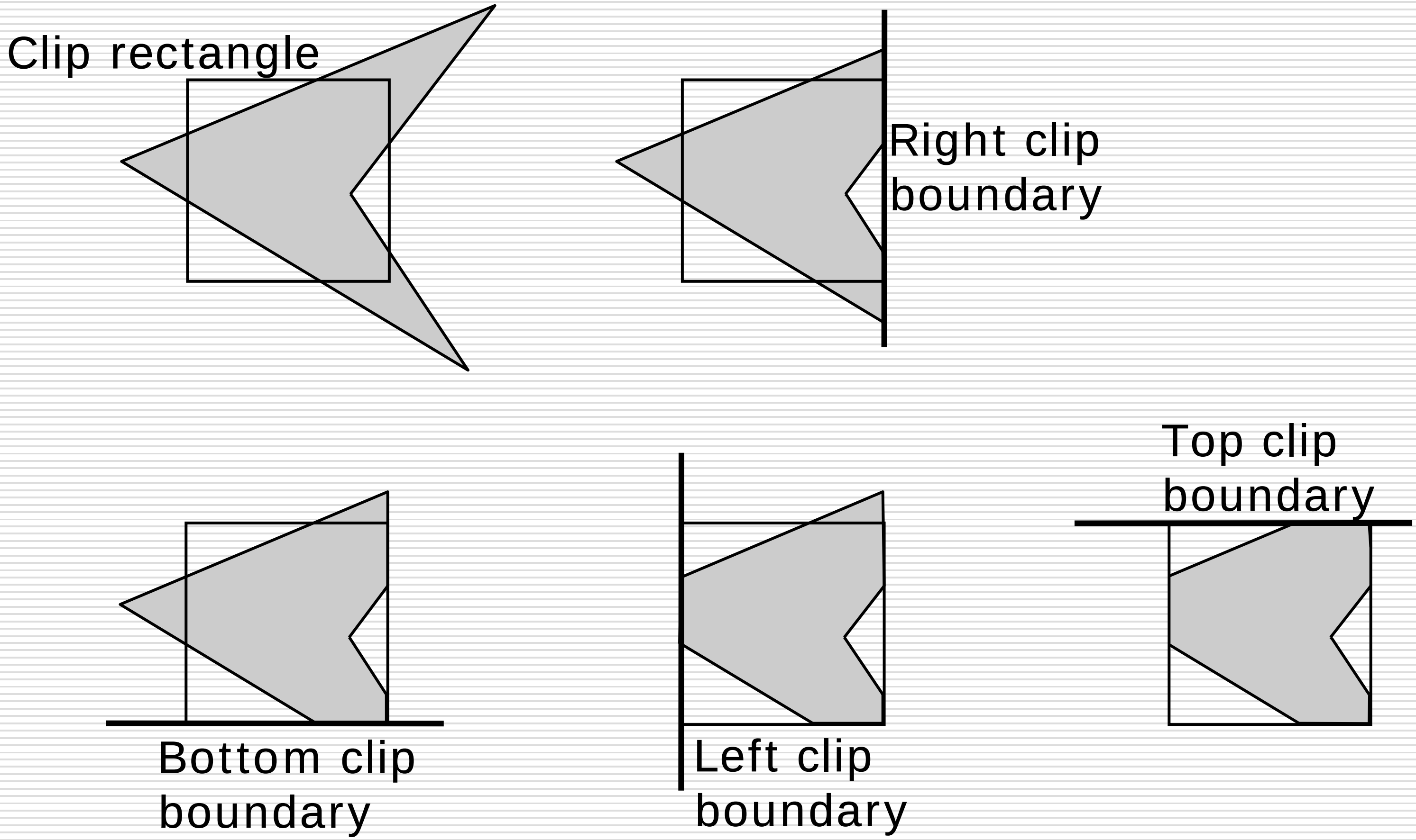
The Cohen-Sutherland Line-Clipping Algorithm



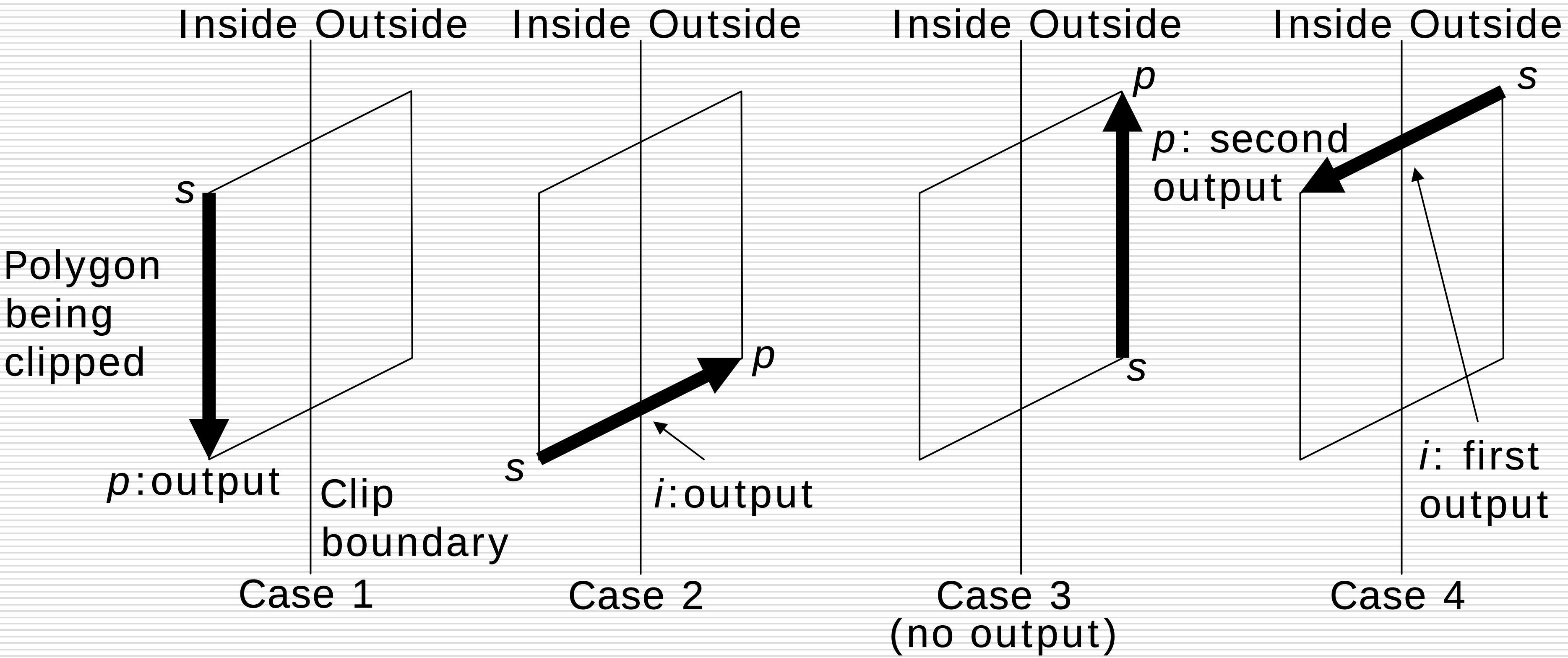
Clipping Polygons



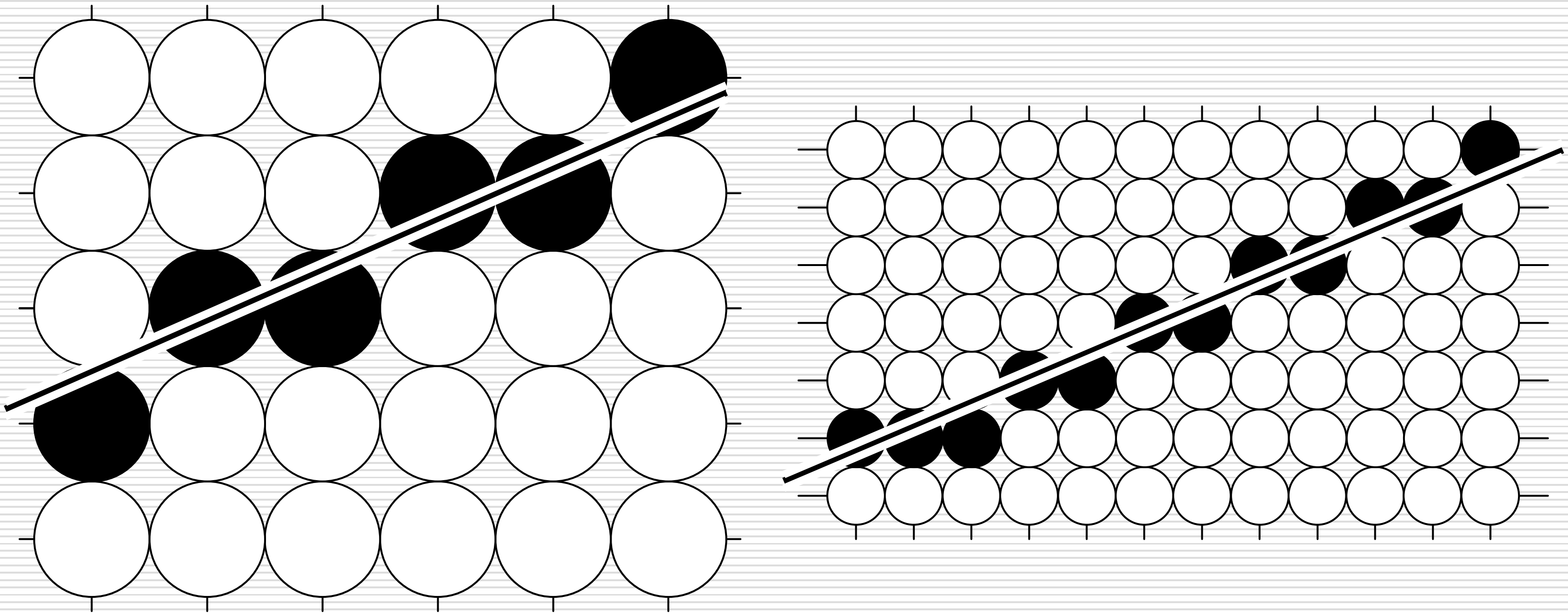
The Sutherland-Hodgman Polygon-Clipping Algorithm



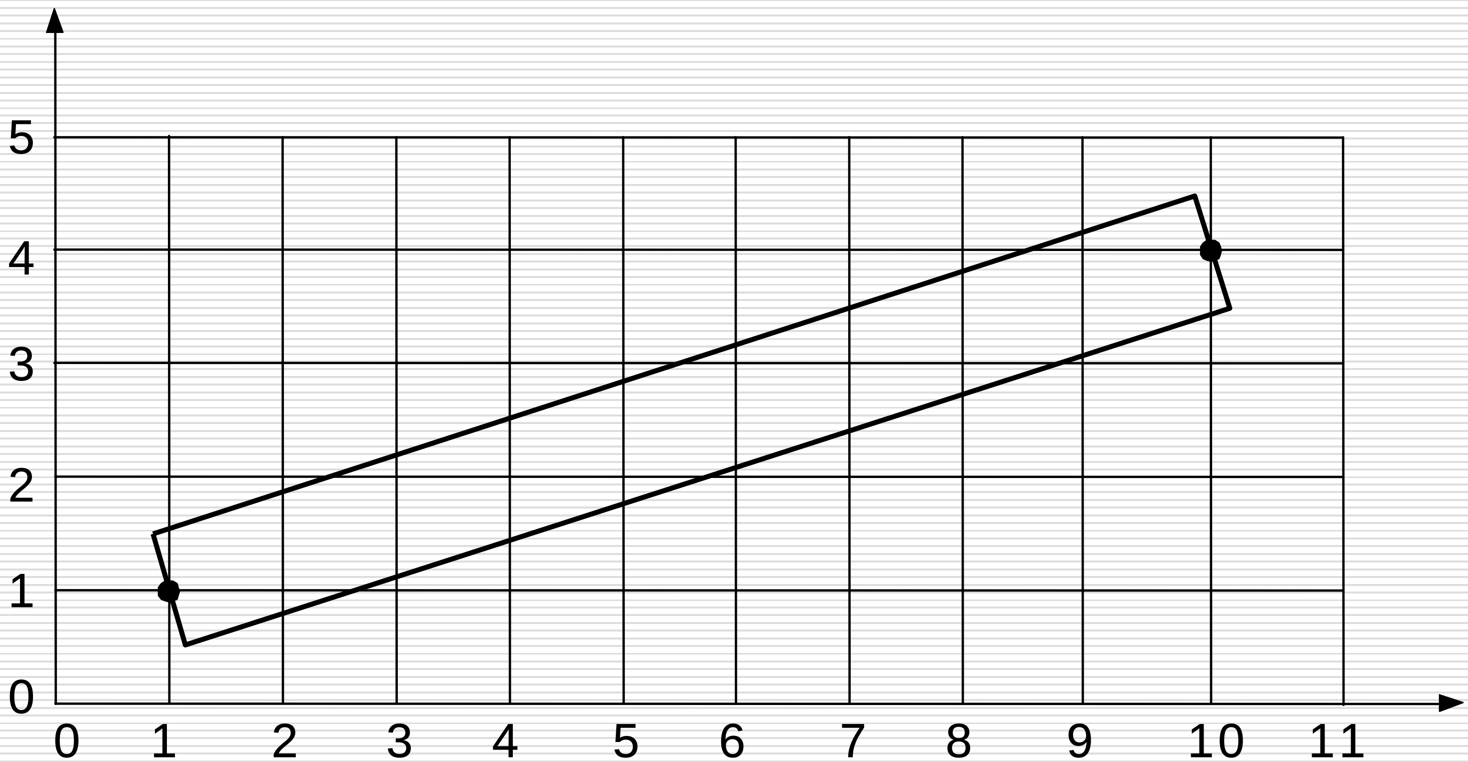
The Sutherland-Hodgman Polygon-Clipping Algorithm



Antialiasing



Unweighted Area Sampling



Unweighted Area Sampling