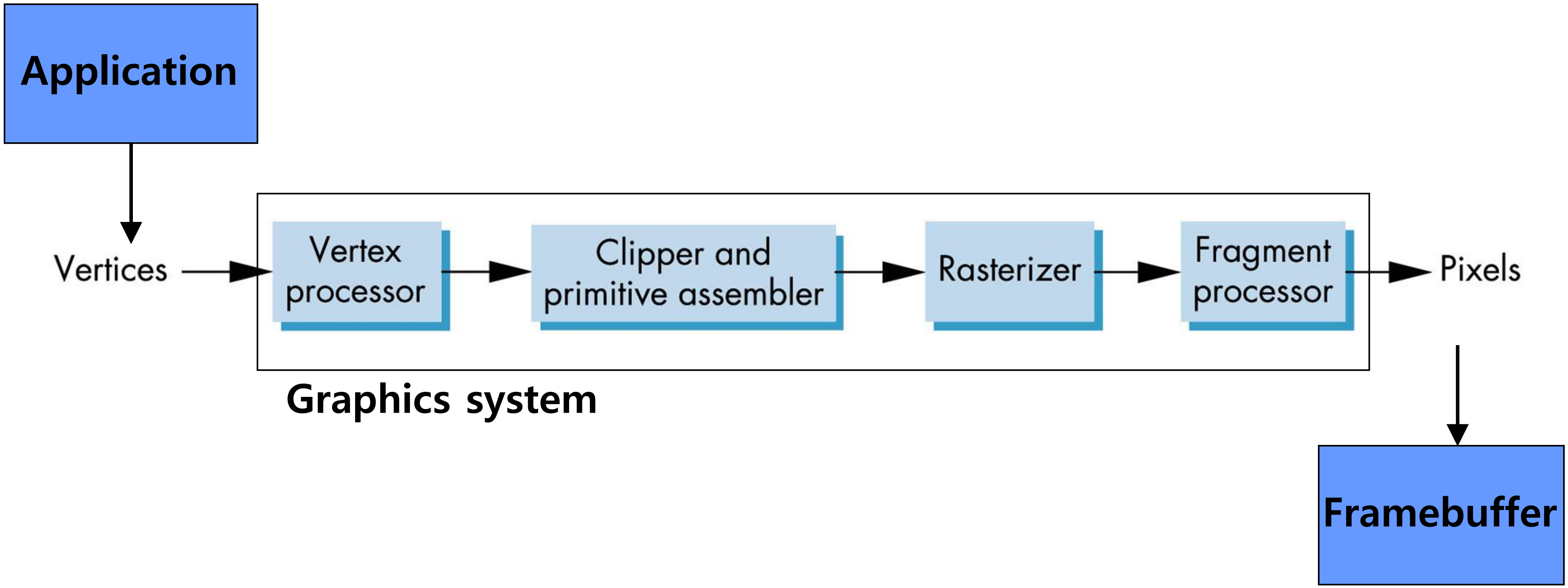
From Vertices to Fragments

Fall 2023
11/30/2023
Kyoung Shin Park
Computer Engineering
Dankook University

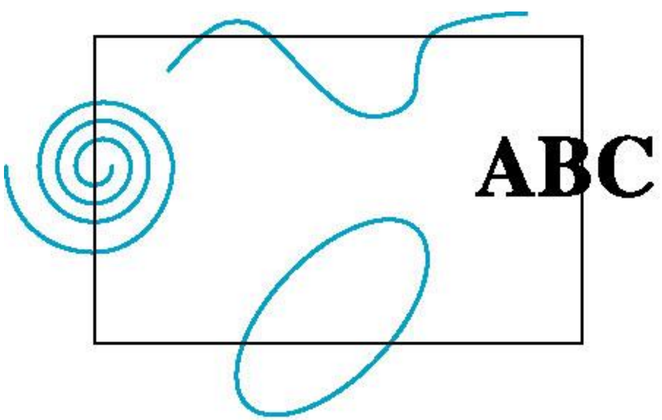
Geometric Pipeline

- Geometric pipeline
  - Vertex processing
  - Clipping and primitive assembly
  - Rasterization
  - Fragment processing



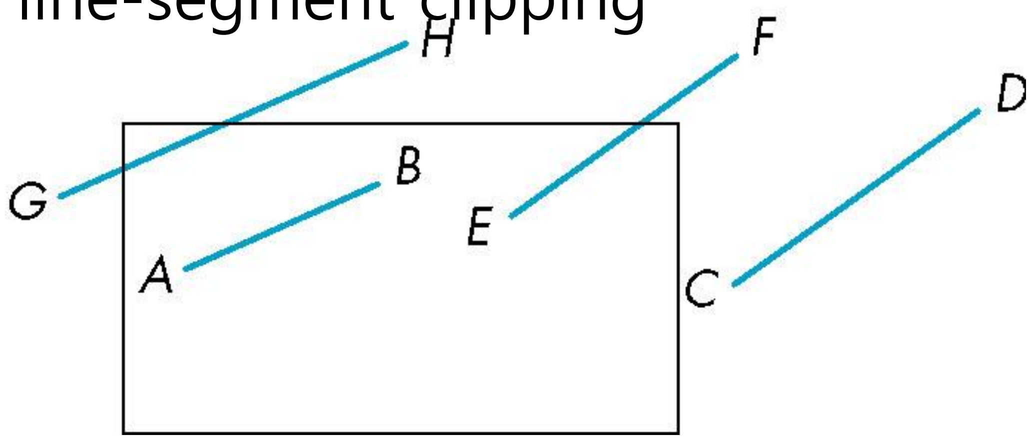
Clipping

- Clipping window
- 3D clipping volume
- Curves and text will be converted to lines and polygons.



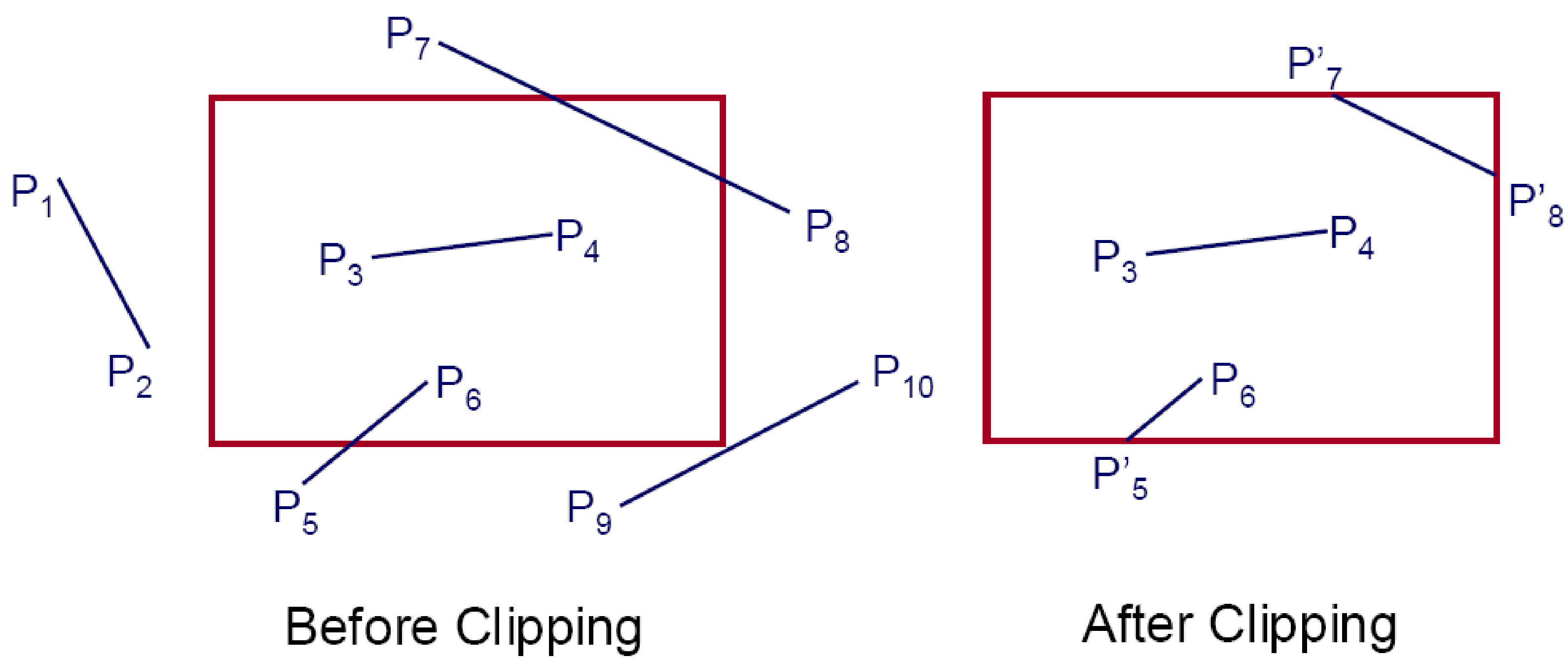
2D Line-Segment Clipping

- Clipping 2D line segments
  - The clipper determines which basic elements or parts of them should appear on the screen and be sent to the rasterizer.
  - Accepted: Basic elements entering the designated viewing space area accepted.
  - Rejected or culled: Basic elements that cannot appear on the screen are removed.
- 2D line-segment clipping



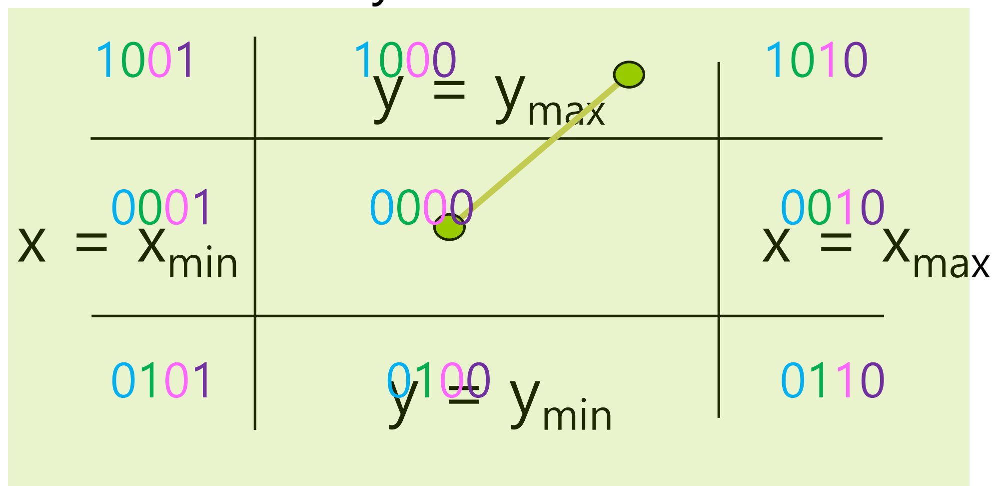
2D Line-Segment Clipping

- How to calculate intersection for all sides of clipping window
  - Inefficient because one division must be performed per intersection



Cohen-Sutherland Algorithm

- Cohen-Sutherland clipping algorithm
  1. Extends the clipping window to infinity on 4 sides and divides the space into 9 areas



  2. Assign a unique **outcode ($b_0b_1b_2b_3$)** to each area as follows.

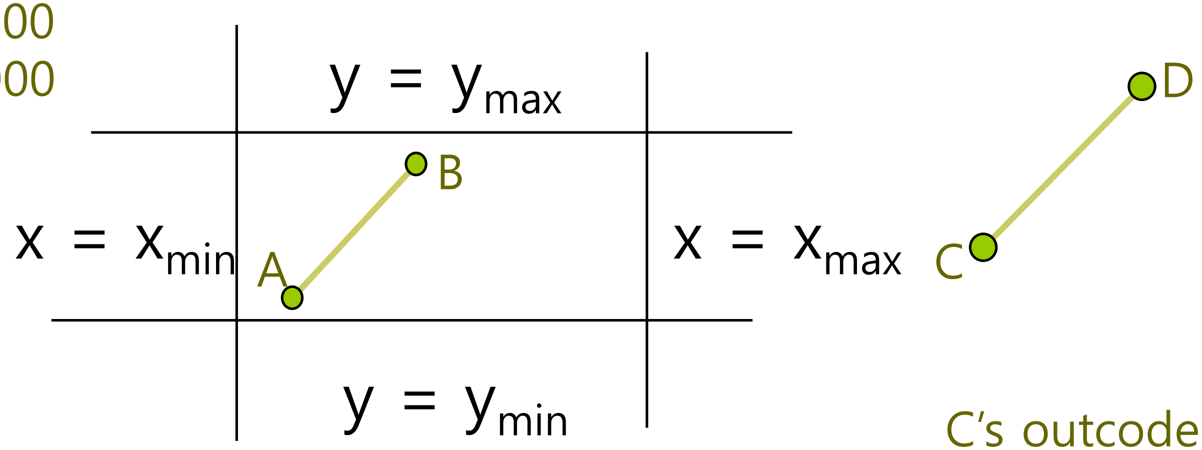
$$b_0 = \begin{cases} 1 & if\ y > y_{max} \\ 0 & otherwise \end{cases} \quad b_1 = \begin{cases} 1 & if\ y < y_{min} \\ 0 & otherwise \end{cases} \quad b_2 = \begin{cases} 1 & if\ x > x_{max} \\ 0 & otherwise \end{cases} \quad b_3 = \begin{cases} 1 & if\ x < x_{min} \\ 0 & otherwise \end{cases}$$

  3. 4 cases are judged based on the outcode.

Cohen-Sutherland Algorithm

- For line segment AB: A's outcode = B's outcode = 0
  - If both ends of the segmented are inside, accepted
- For line segment CD: C's outcode AND D's outcode ≠ 0
  - If both endpoints of the segment are outside the same side of the clipping window, rejected

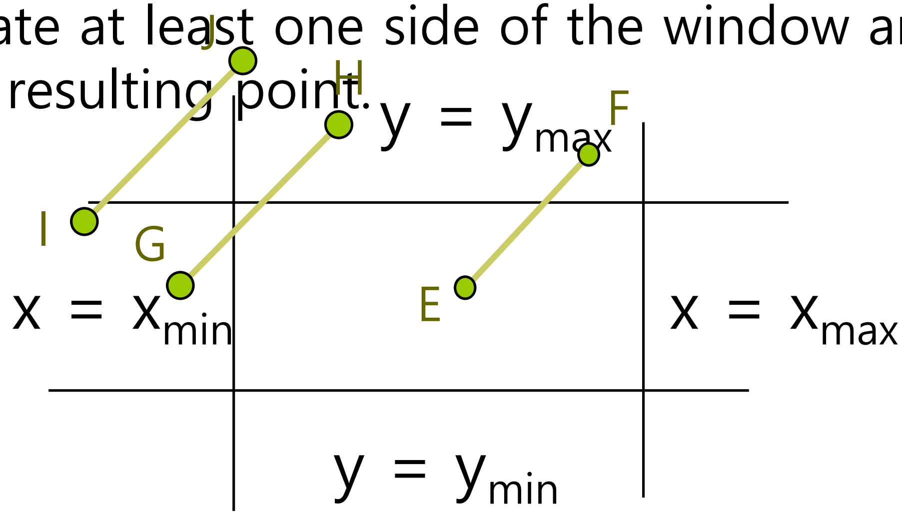
A's outcode = 0000
B's outcode = 0000



C's outcode = 0010
D's outcode = 1010
C AND D = 0010 ≠ 0

Cohen-Sutherland Algorithm

- For line segment EF: E's outcode ≠ 0, F's outcode = 0
  - If one endpoint of the segment is inside the clipping window and the other is outside, subdivide
  - Need to find 1 intersection
- For line segment GH, IJ: G's outcode AND H's outcode = 0
  - If both endpoints of the segment are outside, subdivide. In case of line segment GH, part of the line segment is inside the clipping window.
  - Calculate at least one side of the window and check the outer sign of the resulting point.



G's outcode = 0001
H's outcode = 1000
G AND H = 0000

I's outcode = 0001
J's outcode = 1000
I AND J = 0000

E's outcode = 0000
F's outcode = 1000

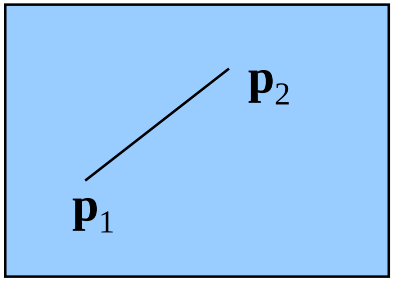
Liang-Barsky Algorithm

- Liang-Barsky clipping algorithm
  1. Parametric line formula

     $$P(\alpha) = (1-\alpha)P_1 + \alpha P_2,\ 0 \le \alpha \le 1$$

     $$x(\alpha) = (1-\alpha)x_1 + \alpha x_2$$

     $$y(\alpha) = (1-\alpha)y_1 + \alpha y_2$$



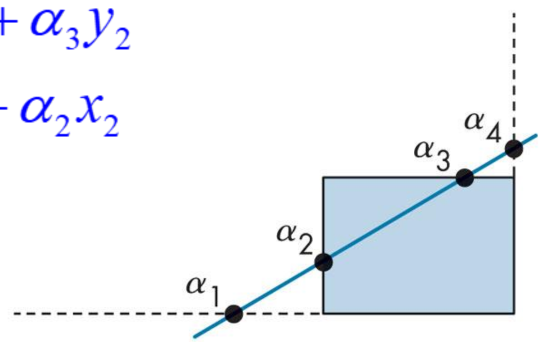
  2. Determined by examining the order of α values by calculating 4 points where the line segment intersects the extended side of the clipping window.

$$y_{\max} = (1-\alpha_3)y_1 + \alpha_3 y_2$$

$$x_{\min} = (1-\alpha_2)x_1 + \alpha_2 x_2$$

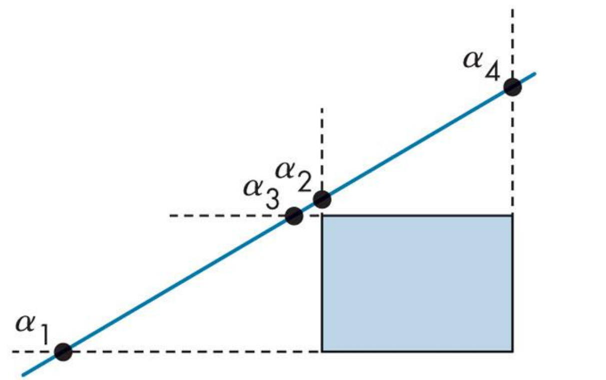
$$\alpha_3 = \frac{y_{\max} - y_1}{y_2 - y_1}$$

$$\alpha_2 = \frac{x_{\min} - x_1}{x_2 - x_1}$$



$$1 > \alpha_4 > \boxed{\alpha_3 > \alpha_2} > \alpha_1 > 0$$

right, top, left, bottom order intersect



$$1 > \underline{\alpha_4 > \alpha_2} > \underline{\alpha_3 > \alpha_1} > 0$$

right, left, top, bottom order intersect

Liang-Barsky Algorithm

- Liang-Barsky clipping algorithm
  3. The line in the clipping window satisfies the following

$$x_{\min} \le x(\alpha) \le x_{\max}$$

$$y_{\min} \le y(\alpha) \le y_{\max}$$

  4. A line outside the clipping window is when (x_1, y_1) is outside x_{min}, x_{max} or y_{min}, y_{max}.

$$q_k < 0 \quad (k = 1,2,3,4)$$

$$\textit{where} \quad q_1 = x_1 - x_{\min}$$

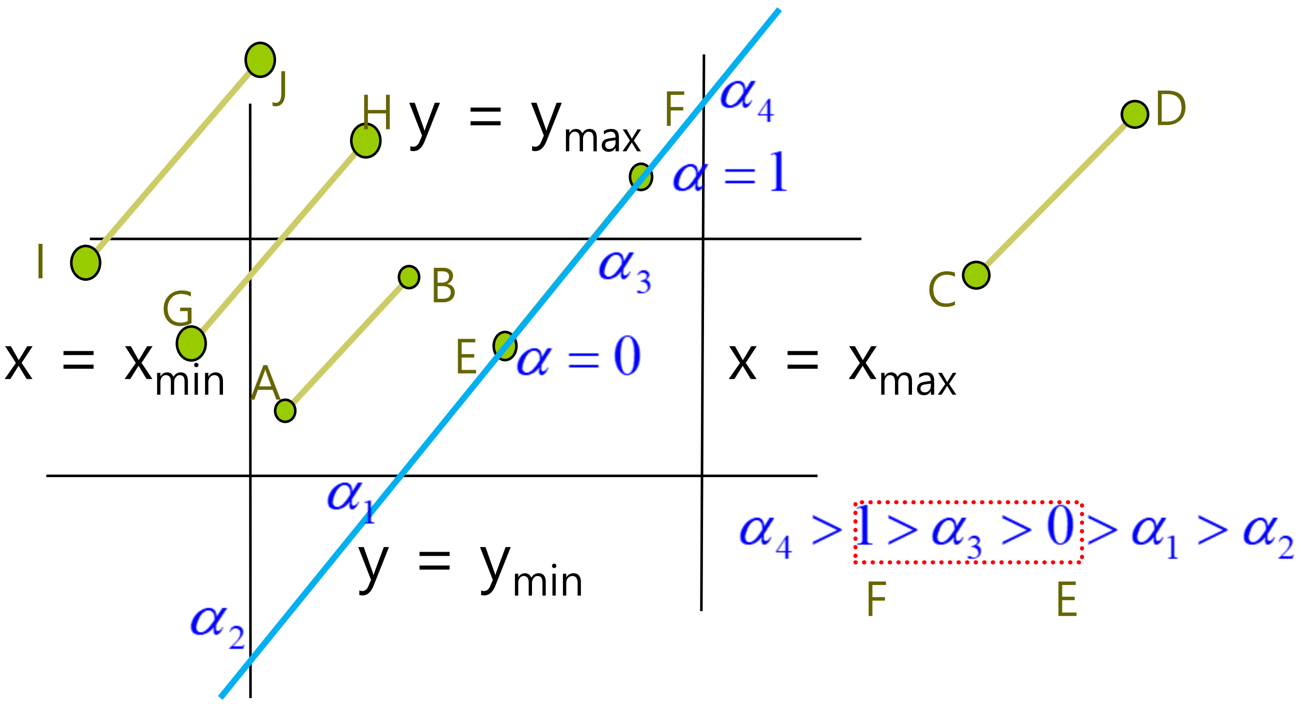
$$q_2 = x_{\max} - x_1$$

$$q_3 = y_1 - y_{\min}$$

$$q_4 = y_{\max} - y_1$$

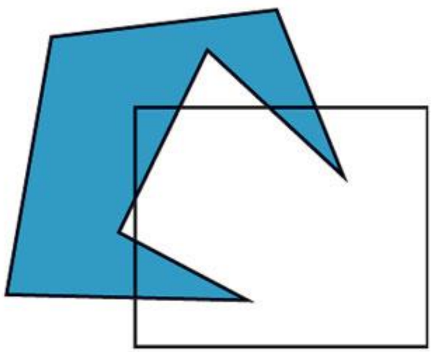
Liang-Barsky Algorithm

- Liang-Barsky clipping algorithm
  5. Of the two points of a straight line, the point with the smallest x value is assumed to be (x_1, y_1). If the line is extended infinitely, the clipping window passes from outside to inside and from inside to outside.

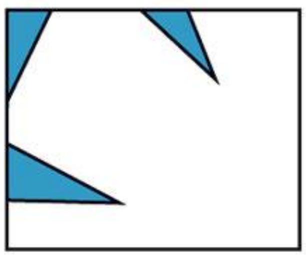


Polygon Clipping

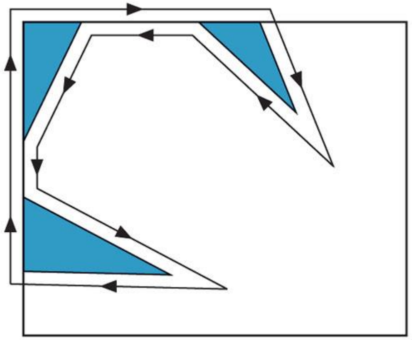
- Concave polygon clipping
  - Method1: How to combine into one polygon after clipping
  - Method2: Split into a set of concave polygons (tessellate), and clipping



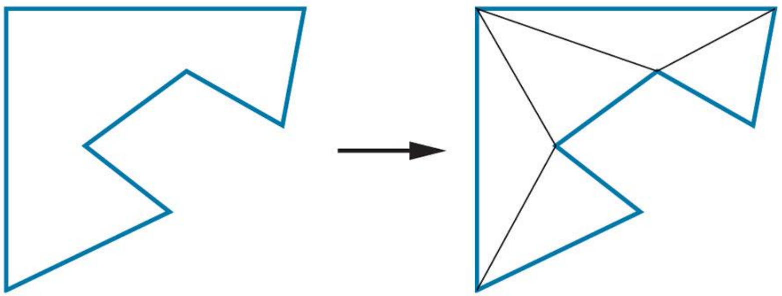
Before clipping



After clipping



Create one polygon



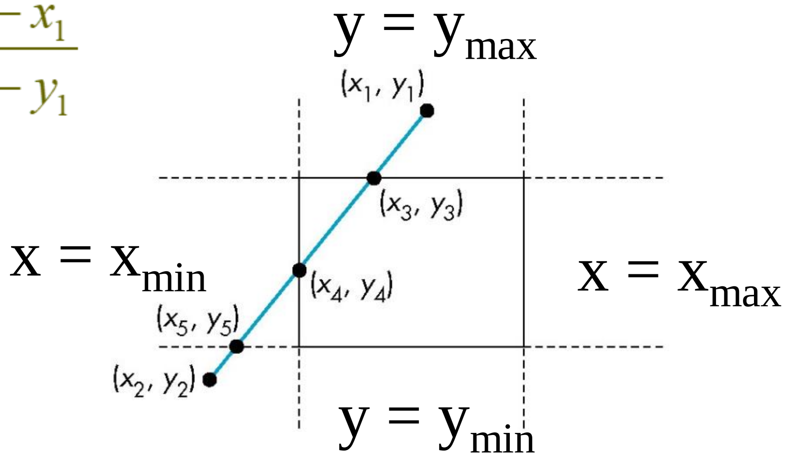
Tessellation

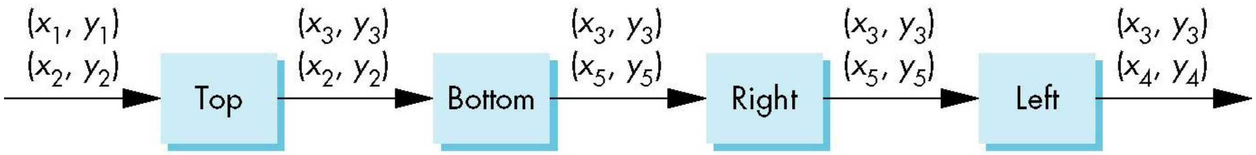
Pipeline Clipping of Line Segments

- Sutherland-Hodgeman algorithm
  - Subdividing the cutter into a simpler cutter pipeline that clips each side of the window.

$$x_3 = x_1 + (y_{max} - y_1)\frac{x_2 - x_1}{y_2 - y_1}$$

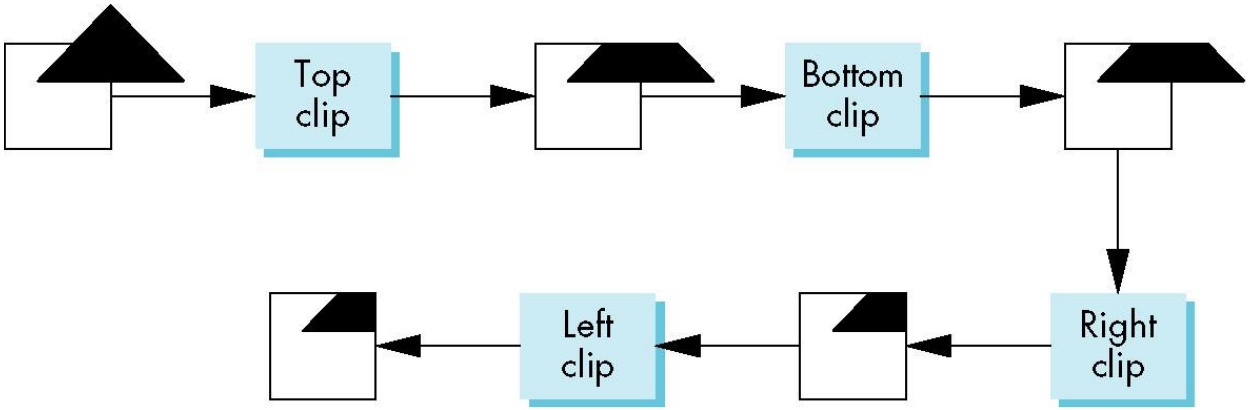
$$y_3 = y_{max}$$





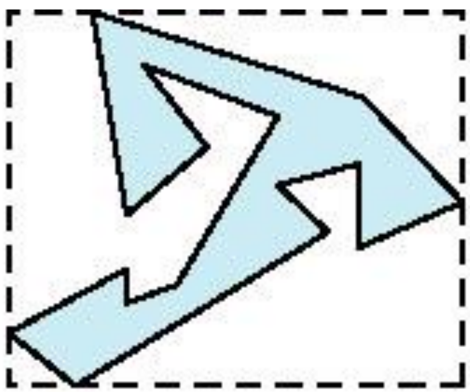
Pipeline Clipping of Polygons

- Sutherland-Hodgeman algorithm
  - Input: Polygon (vertices list) and clipping plane
  - Output: New clipped polygon (vertices list)
  - For 2D, pipeline clipping of polygons
  - For 3D, add front and back clipping



Bounding Boxes

- Use the *axis-aligned bounding box* or *extent of a polygon* for clipping
  - For complex polygons with many sides
  - Bounding box is the smallest rectangle aligned to the window containing the polygon
  - The bounding box is obtained by calculating the minimum (min) and maximum (max) values of the x and y values of the polygon vertices.



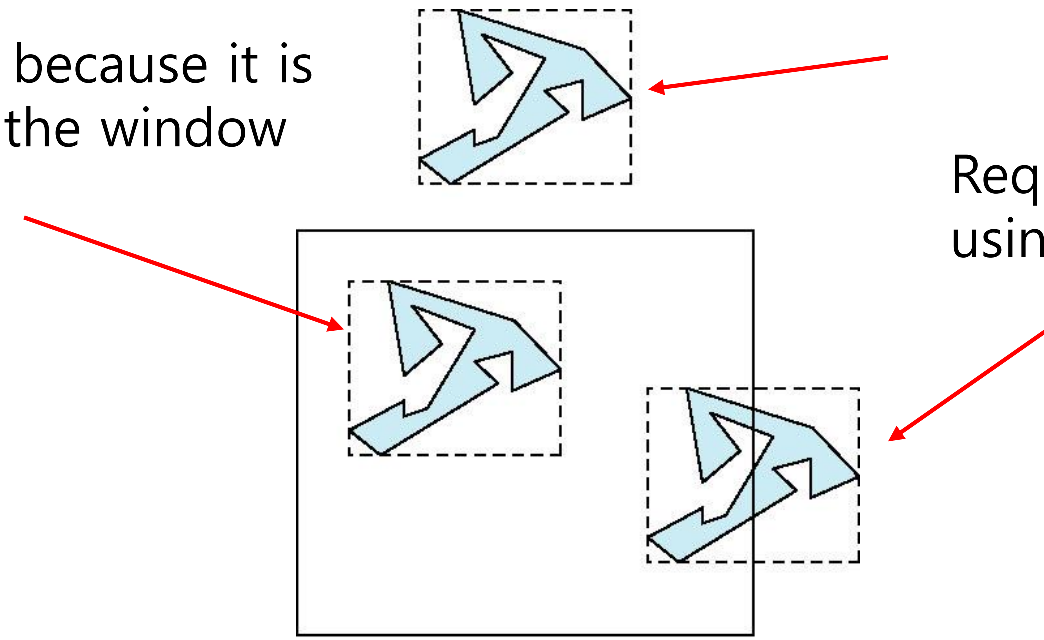
Bounding boxes

- Simple clipping using bounding boxes

Reject, because it is
outside of the window

Accept, because it is
inside the window

Requires detailed clipping
using all sides of the polygon

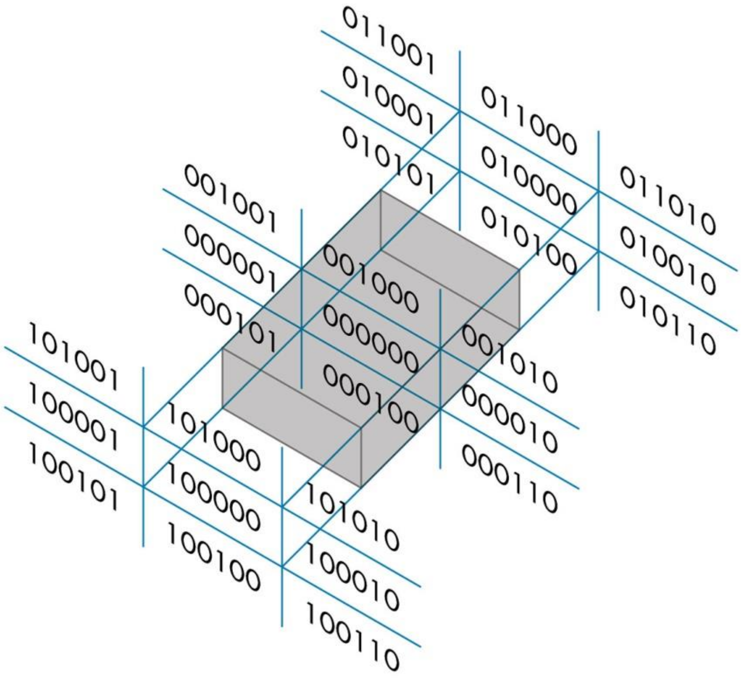


Cohen-Sutherland Algorithm in 3D

- In 3D, clipping for the bounding volume, not the bounding area
- Cohen-Sutherland clipping algorithm
  - Calculate using 6-bit outcode in 3D (instead of 4-bit outcode used in 2D)

$$b_4 = \begin{cases} 1 & if\ z > z_{max} \\ 0 & otherwise \end{cases}$$

$$b_5 = \begin{cases} 1 & if\ z < z_{min} \\ 0 & otherwise \end{cases}$$



Liang-Barsky Algorithm in 3D

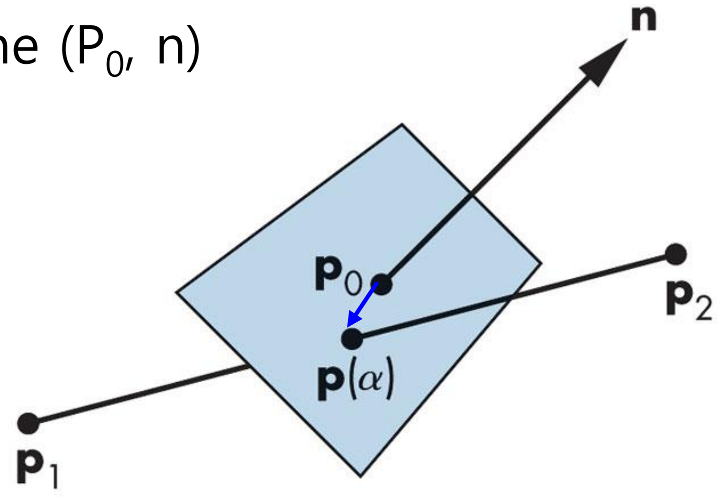
- Liang-Barsky clipping algorithm
  - 3D Line parametric form

$$P(\alpha) = (1-\alpha)P_1 + \alpha P_2,\ 0 \le \alpha \le 1$$
$$x(\alpha) = (1-\alpha)x_1 + \alpha x_2$$
$$y(\alpha) = (1-\alpha)y_1 + \alpha y_2$$
$$z(\alpha) = (1-\alpha)z_1 + \alpha z_2$$

  - Derive α from the formula of plane (P_0, n)

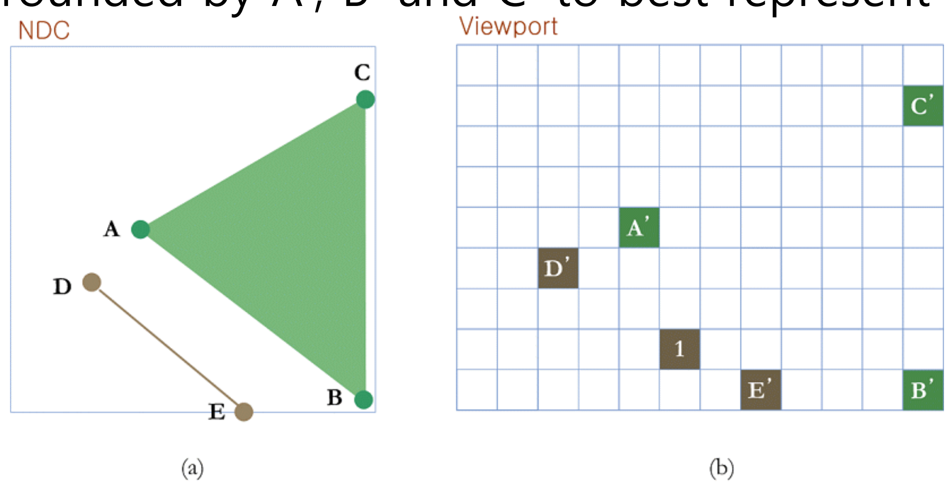
$$P(\alpha) = (1-\alpha)P_1 + \alpha P_2$$
$$n \cdot (P(\alpha) - P_0) = 0$$

$$\alpha = \frac{n \cdot (P_0 - P_1)}{n \cdot (P_2 - P_1)}$$



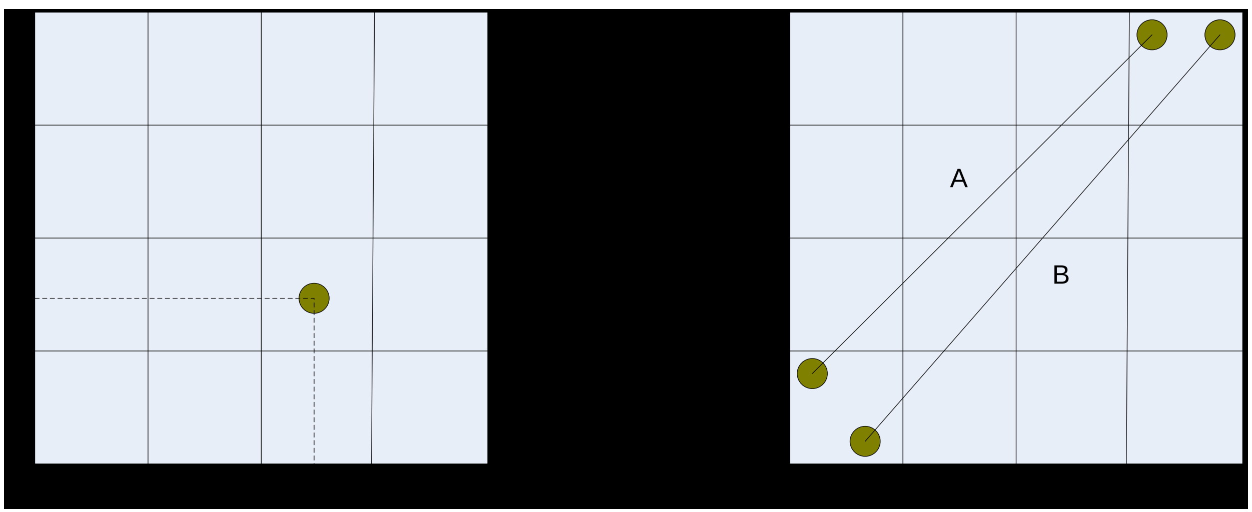
Rasterization

- Rasterization/Scan conversion
  - The final step in the process from framebuffer to fragment
  - The task of deciding which pixels to represent an object
  - Mapping from normalized device coordinates to viewport
    - Based on the result of converting vertex coordinates to screen coordinates
    - Convert line segment to screen coordinates
    - Convert inner surface to screen coordinates
    - In the picture below, what pixels should be painted in the area surrounded by A′, B′ and C′ to best represent the triangle ABC?



Rasterization

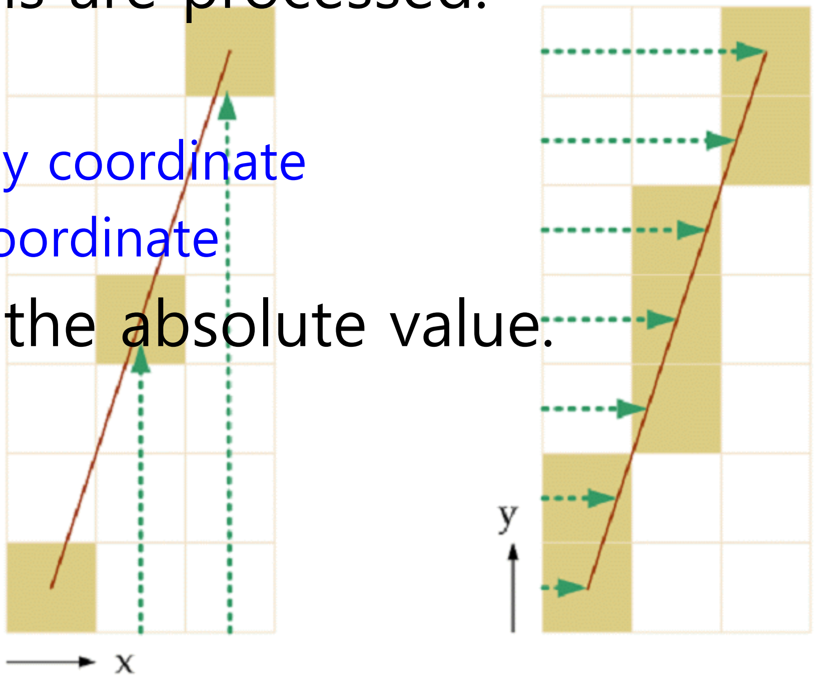
- Convert float coordinates to integer coordinates
  - Sometimes, rounding is necessary.
  - For example, convert the vertex's viewpoint coordinates (1.95, 1.4) → pixel (2, 1)
  - All vertices that are (1.5 <= x < 2.5) and (0.5 <= y < 1.5) inside the boundary are mapped to (2, 1)



A and B are all mapped to the same line segment.

Line Scan-Conversion

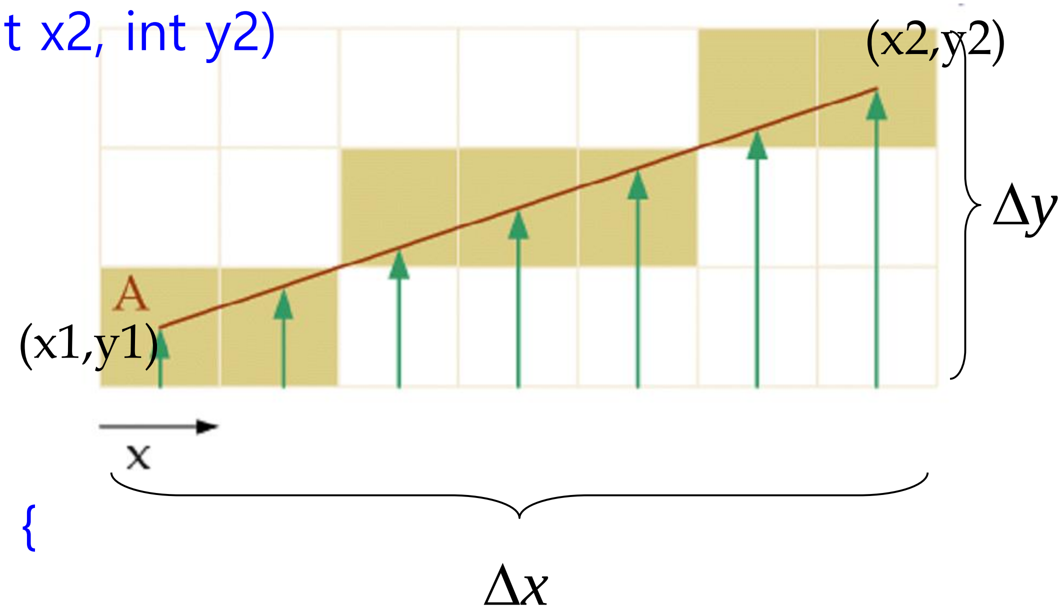
- A line segment is the most primitive to which the rasterization algorithm that is applied.
- Once the vertices at both ends of the segment have been determined to which pixels on the screen are mapped, the remaining pixels are processed.
- Sampling by slope
  - If greater than 1, increase the y coordinate
  - If less than 1, increase the x coordinate
- If the slope is negative, use the absolute value.



Line Scan-Conversion

- The following line scan-conversion equation is slow due to floating point multiplication.

```
void LineDraw(int x1, int y1, int x2, int y2)
{
    float y, m;
    int dx, dy;
    dx = x2 - x1;
    dy = y2 - y1;
    m = dy / dx;
    for (x = x1; x <= x2; x++)  {
        y = m*(x - x1) + y1;
        DrawPixel(x, round(y));
    }
}
```



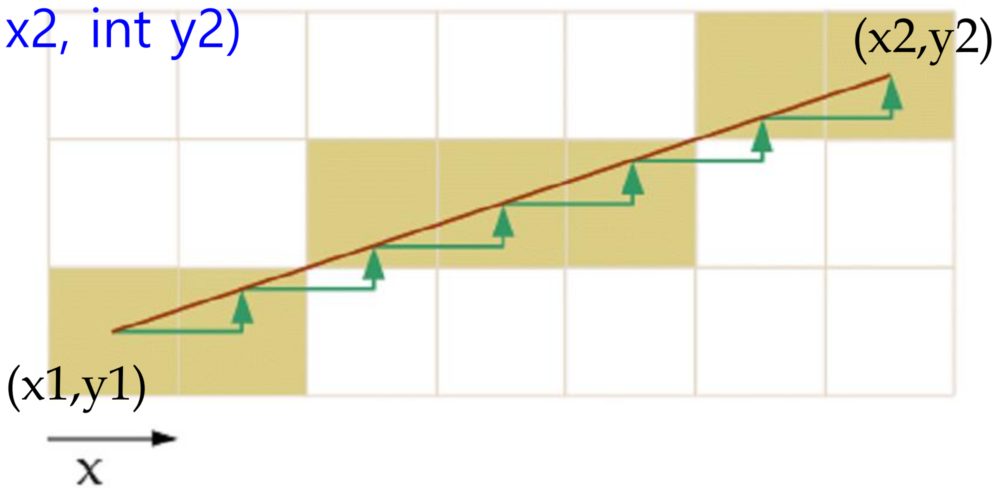
두 점 $(x1, y1)(x2, y2)$을 지나는 직선 방정식

$$y = \frac{y2 - y1}{x2 - x1}(x - x1) + y1$$

DDA (Digital Differential Analyzer)

- The following line scan-conversion equation converts floating-point multiplication to floating-point addition

```
void LineDraw(int x1, int y1, int x2, int y2)
{
    float m, y;
    int dx, dy;
    dx = x2 - x1;
    dy = y2 - y1;
    m = dy / dx;
    y = y1;
    for (int x = x1; x <= x2; x++) {
        y += m;
        DrawPixel(x, round(y));
    }
}
```



$$y = mx + h \ \text{where} \ m = \frac{y2 - y1}{x2 - x1} = \frac{\Delta y}{\Delta x}$$

$$\Rightarrow \Delta y = m \Delta x$$

$$\Rightarrow \Delta y = m \ (\text{x가 1씩 증가할 때})$$

DDA (Digital Differential Analyzer)

- DDA algorithm

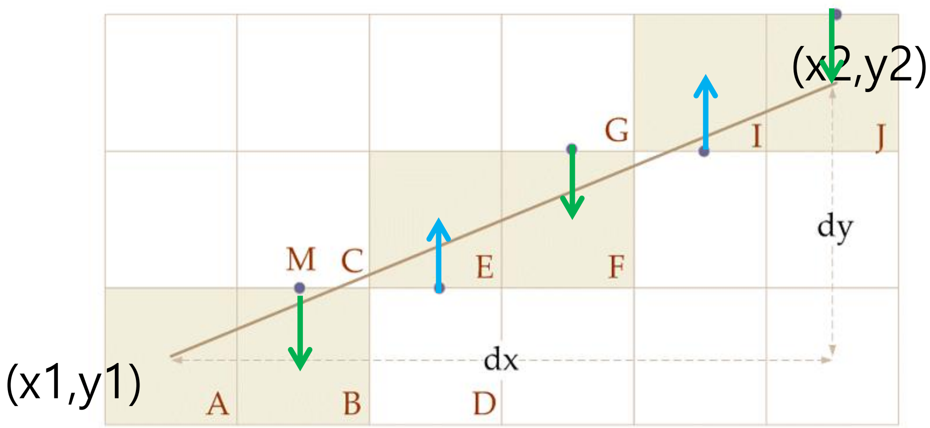
x	(x, y)	반올림 결과
x = 0	(0, 0.00)	(0, 0)
x = 1	(1, 0.33)	(1, 0)
x = 2	(2, 0.66)	(2, 1)
x = 3	(3, 0.99)	(3, 1)
x = 4	(4, 1.32)	(4, 1)
x = 5	(5, 1.65)	(5, 2)
x = 6	(6, 1.98)	(6, 2)

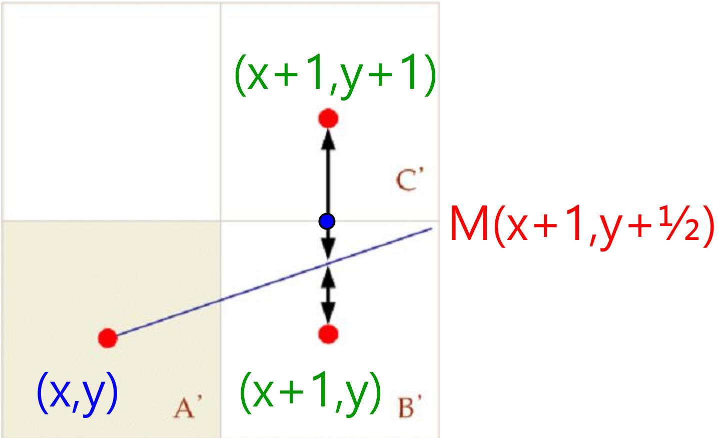
DDA (Digital Differential Analyzer)

- DDA disadvantage
  - Floating-point arithmetic operation
    - Floating-point addition is slower than integer arithmetic operation
  - Rounding
    - Time it takes to execute the round( ) function
  - Accuracy
    - In case of floating point numbers, the back seat is cut off
    - Accumulation of errors by successive addition
    - Selected pixels gradually move away from the actual line segment and thus drift

Bresenham's Line Algorithm

- Also known as Midpoint Algorithm
  - Avoid all floating point calculations and use only integer.
  - The line rasterization algorithm, the standard for raster machines.





- Select A (x, y)
  - The next pixel is one of B (x+1, y), or C (x+1, y+1)
  - Determined by the vertical distance between the center of the pixel and the line segment
  - **Select Pixel B if the segment is below the midpoint M, pixel C if it is above.**

Bresenham's Line Algorithm

- If pixel A=(x1, y1), the coordinates of the midpoint M of pixel B and C are (x1 + 1, y1 + ½), substituting this into F:

$$y = mx + h, m = \frac{dy}{dx}$$

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

$$ydx = xdy + hdx$$

$$0 = xdy - ydx + hdx$$

$$F(x, y) = 2xdy - 2ydx + 2hdx$$

$$F(x, y) = F\left(x1 + 1, y1 + \frac{1}{2}\right)$$

$$= 2(x1 + 1)dy - 2\left(y1 + \frac{1}{2}\right)dx + 2hdx$$

$$= 2x1dy - 2y1dx + 2hdx + 2dy - dx$$

$$= F(x1, y1) + 2dy - dx$$

$$F(x1, y1) = 2x1dy - 2y1dx + 2hdx = 0$$

$$F(x, y) = 2dy - dx$$

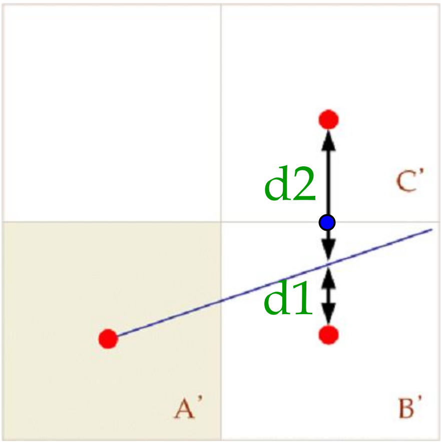
Bresenham's Line Algorithm

- Determine whether the midpoint is above or below the line segment based on the **decision variable, F**.
  - If F(x, y) < 0, the midpoint is on the line segment and therefore selects the **East** pixel.
  - If F(x, y) > 0, select the **NorthEast** pixel.

$$F(x,y) = 2dy - dx$$

$$if\,(F(x,y) < 0) \quad select\ E \quad //\text{동쪽 화소 선택}$$

$$else \quad select\ NE \quad //\text{동북쪽 화소 선택}$$



d2>d1 => F(x,y) <0

Bresenham's Line Algorithm

- The current pixel is (x, y) and if the **East** pixel is selected, the next step position is **(x+1, y)**.
- If the **NorthEast** pixel is selected, the next step position is **(x+1, y+1).**
- The difference between the decision variable at the next stage and the decision variable at the current stage is:

$$
\begin{aligned}
incrE &= F(x+1, y) - F(x, y) \\
&= \left(2(x+1)dy - 2ydx + 2hdx\right) - \left(2xdy - 2ydx + 2hdx\right) \\
&= 2dy
\end{aligned}
$$

$$
\begin{aligned}
incrNE &= F(x+1, y+1) - F(x, y) \\
&= \left(2(x+1)dy - 2(y+1)dx + 2hdx\right) - \left(2xdy - 2ydx + 2hdx\right) \\
&= 2dy - 2dx
\end{aligned}
$$

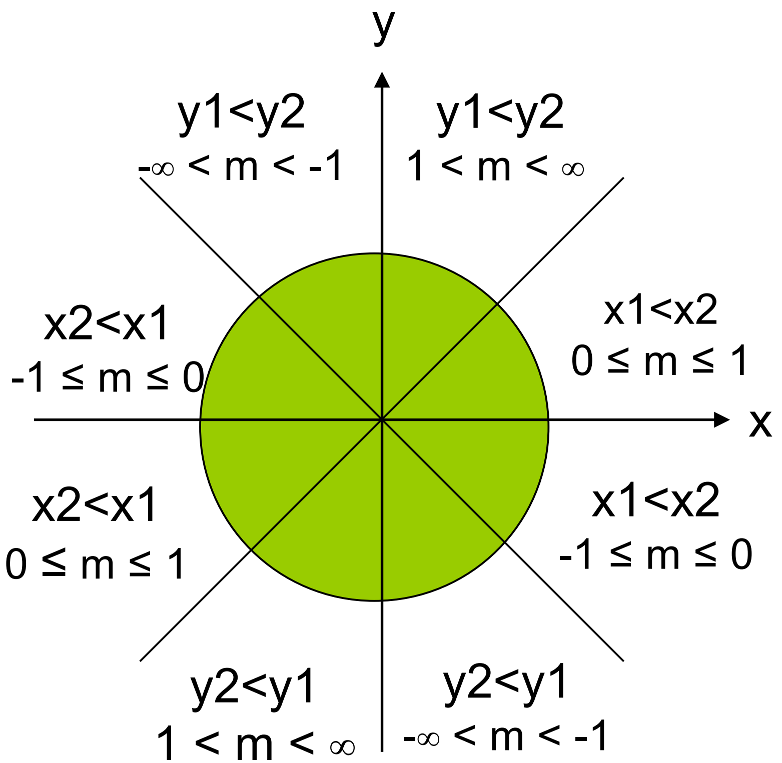
Bresenham's Line Algorithm

0 ≤ m ≤ 1

```
void MidpointLineDraw(int x1, int y1, int x2, int y2)
{
    int dx, dy, incrE, incrNE, D, x, y=y1;
    dx = x2 - x1; dy = y2 - y1;
    D = 2*dy - dx;              // initialize the decision variable
    incrE = 2*dy;               // increment when selecting East
    incrNE = 2*dy - 2*dx;       // increment when selecting NE
    for (x=x1; x <= x2; x++) {
        if (D <= 0) {           // If the decision variable is negative,
            D += incrE;         // select E and increase decision variable
        }
        else {                  // If the decision variable is positive,
            D += incrNE;        // select NE, increase decision variable
            y++;                // y++ next pixel is NE
        }
        DrawPixel (x, y);       // draw pixel
    }
}
```

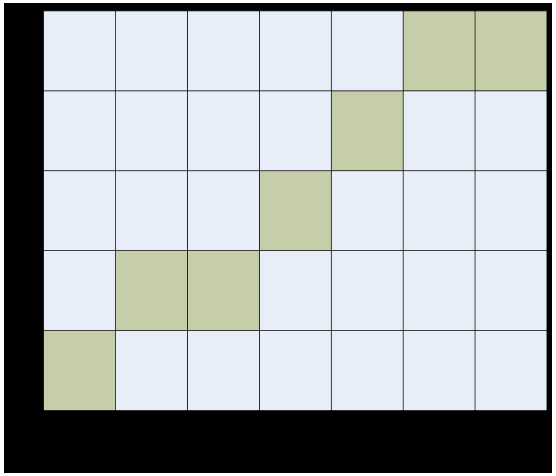
Bresenham's Line Algorithm

- $|m| > 1.0$
  - Calculate by swapping x and y
  - Increasing in the y direction, determine the x-value
- In addition, special cases are handled separately.
  - $\Delta y = 0$ (horizontal line)
  - $\Delta x = 0$ (vertical line)
  - $|\Delta x| = |\Delta y|$ (diagonal lines)



Bresenham's Line Algorithm

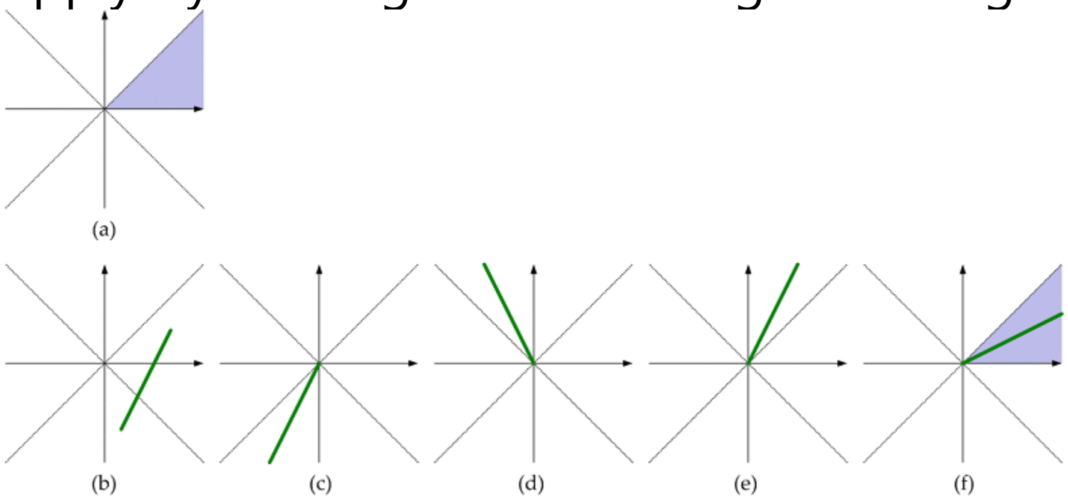
- For example, a line segment between (0, 0) and (6, 4)



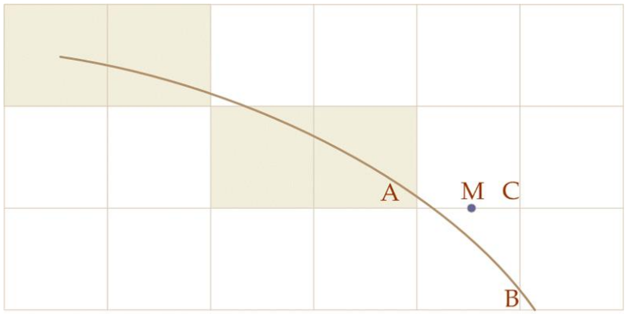
(0, 0)	D > 0
(1,1)	D < 0
(2, 1)	…
…	…
(6, 4)	

Bresenham's Line Algorithm

- Increased speed by integer operation + hardware implementation
- Defined only in the first 8th
  - Apply by moving and reflecting other segments

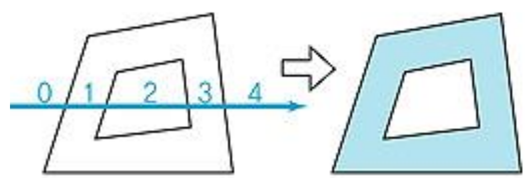


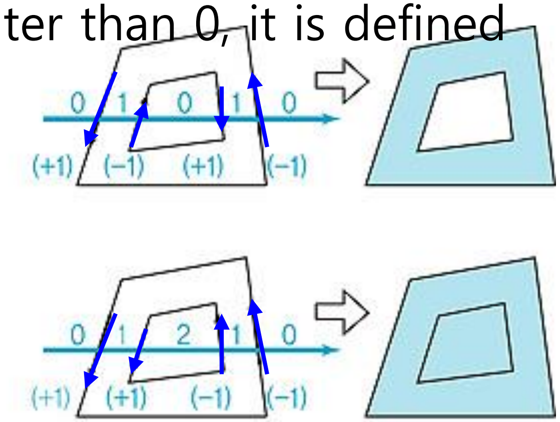
- Circle algorithm
  - Similar to line segment algorithm



Polygon Scan-Conversion

- Polygon rasterization = polygon filling
  - If the point is inside the polygon, paint it with the interior color
- Polygon inside/outside rule
  - Even-odd rule
    - If the boundary of each scan line intersects the odd number, it is inside. If it intersects the even number, it is outside.
  - Non-zero winding rule
    - When each scan line crosses the lower boundary, the number of folds increased by 1, and when it crosses the upper boundary, it is decreased by 1.
    - At this time, if the number of folds is greater than 0, it is defined as the inner area of the polygon.

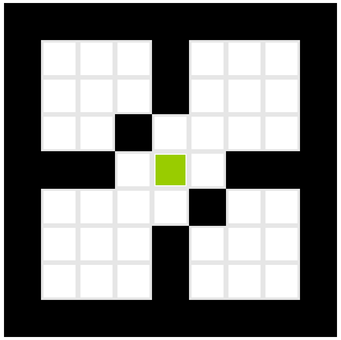




Flood Fill

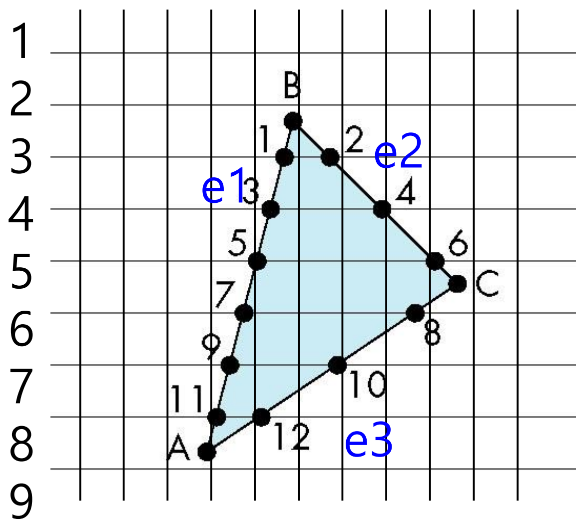
- Flood fill
  - Filling an area defined as interior
  - Starting at the seed point inside the polygon, looping through the neighbors, if they are not side points, paint with a fill color.

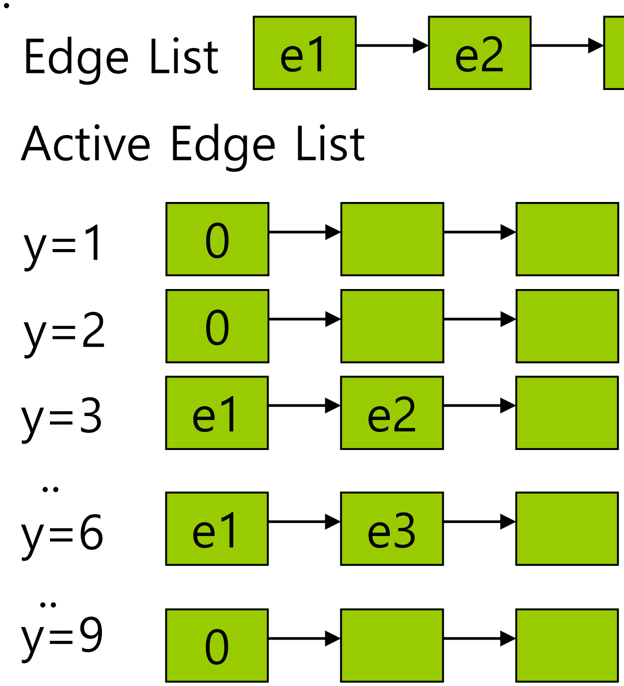
```
void flood_fill(int x, int y) { // Start at the initial point (x, y) inside polygon
    if(read_pixel(x,y)= = WHITE) { // if current pixel is background color
        write_pixel(x,y,BLACK);     // paint with fill color
        flood_fill(x+1, y);          // repeat right side
        flood_fill(x-1, y);         // repeat left side
        flood_fill(x, y+1);         // repeat down side
        flood_fill(x, y-1);         // repeat up side
    }
}
```



Scan Line Fill

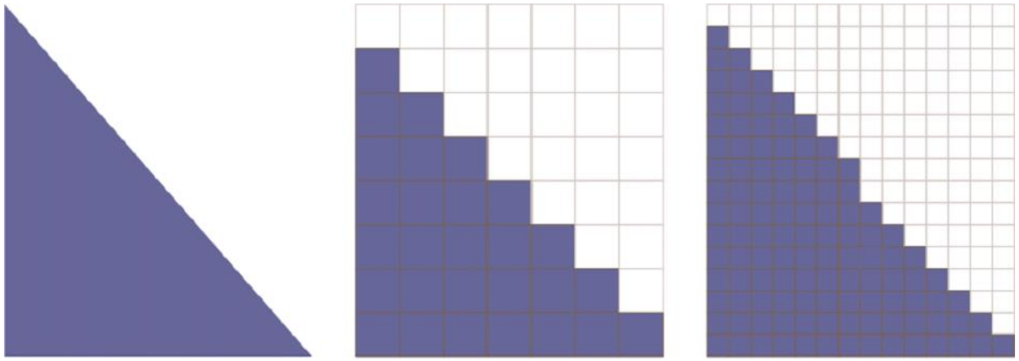
- Scan line fill
  - Y-X polygon scan line algorithm:
    - Compose Edge list (EL) by arranging all edges in Y-value order
    - Take out the edge from EL where each scan line intersects, and move it to the Active Edge List (AEL).
    - Fill the gap b=y pairing the scan line with each edge and intersection point by two.





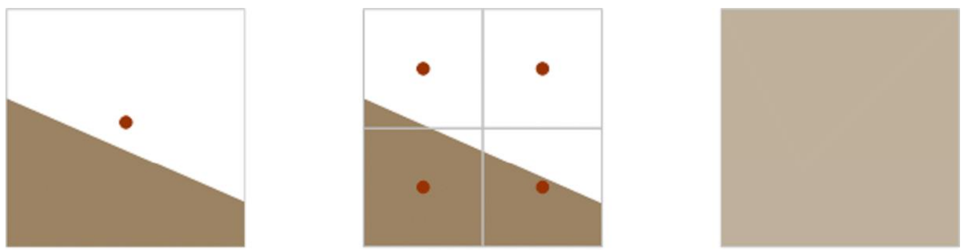
Aliasing

- Stair-step (Jaggies) border
  - In bitmap representation, it is only possible to approximate pixel units.
  - An inevitable phenomenon when an object with infinite resolution is approximated in units of pixel with finite resolution.

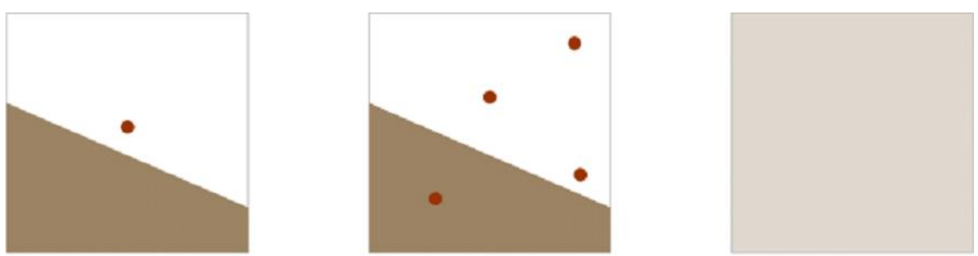


Anti-Aliasing

- Super-Sampling
  - Sampling in partial pixels. Post filtering
  - Reflects the average value of partial pixels

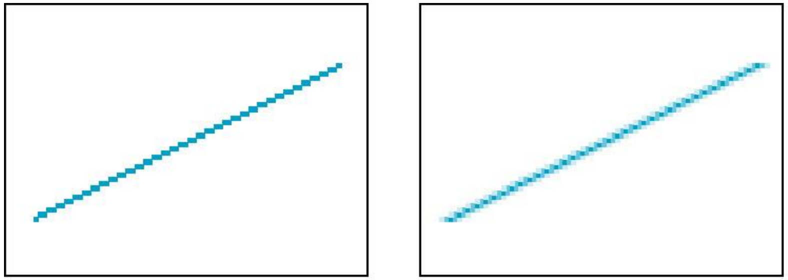


  - Super sampling by jitter
    - If the object itself is irregular, irregular sampling is advantageous.

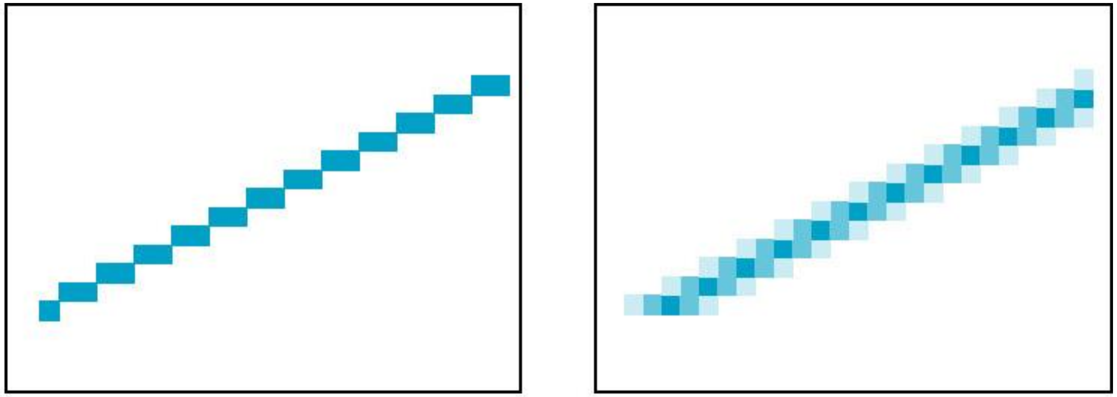


Anti-Aliasing

Aliasing



Anti-aliased



Magnified