2D Viewing

Part II

Clipping Algorithms

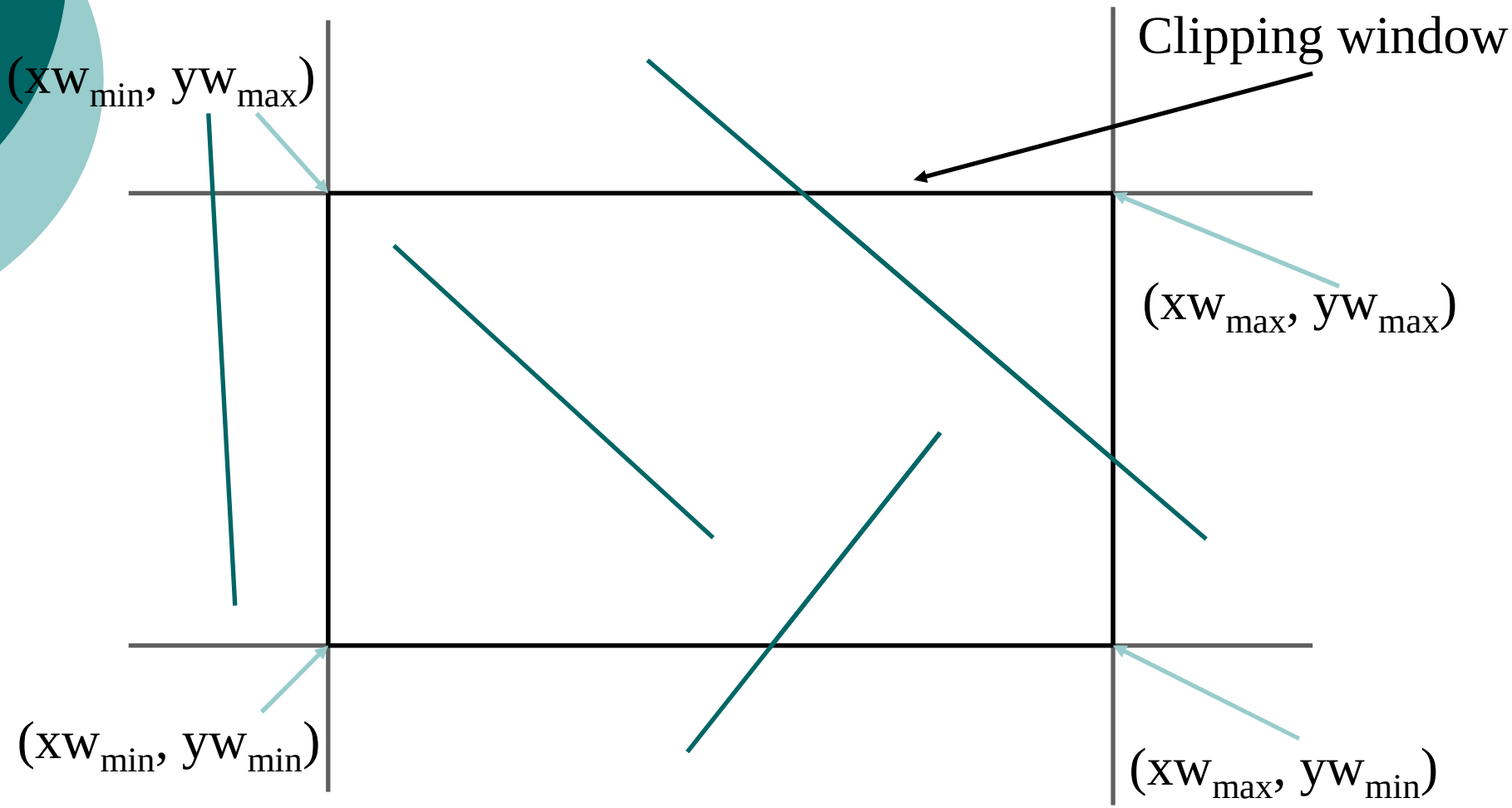
Why would we clip?

- Clip objects before rasterization.
- To avoid unnecessary work:
  - pixel coordinate calculations
  - parameter interpolation

Type of Clipping

- Point Clipping
- **Line Clipping**
- Fill-Area Clipping
- Curve Clipping
- Text Clipping

Line Clipping



Line Clipping Algorithms

- Simple Line Clipping
- Cohen-Sutherland Algorithm
- Liang-Barsky Algorithm

Simple Line Clipping

- Using **point-clipping test** to determine completely INSIDE or OUTSIDE line.

$$xw_{\min} \le x \le xw_{\max}$$

$$yw_{\min} \le y \le yw_{\max}$$

- Using **parametric equations** to determine partially INSIDE or OUTSIDE line.

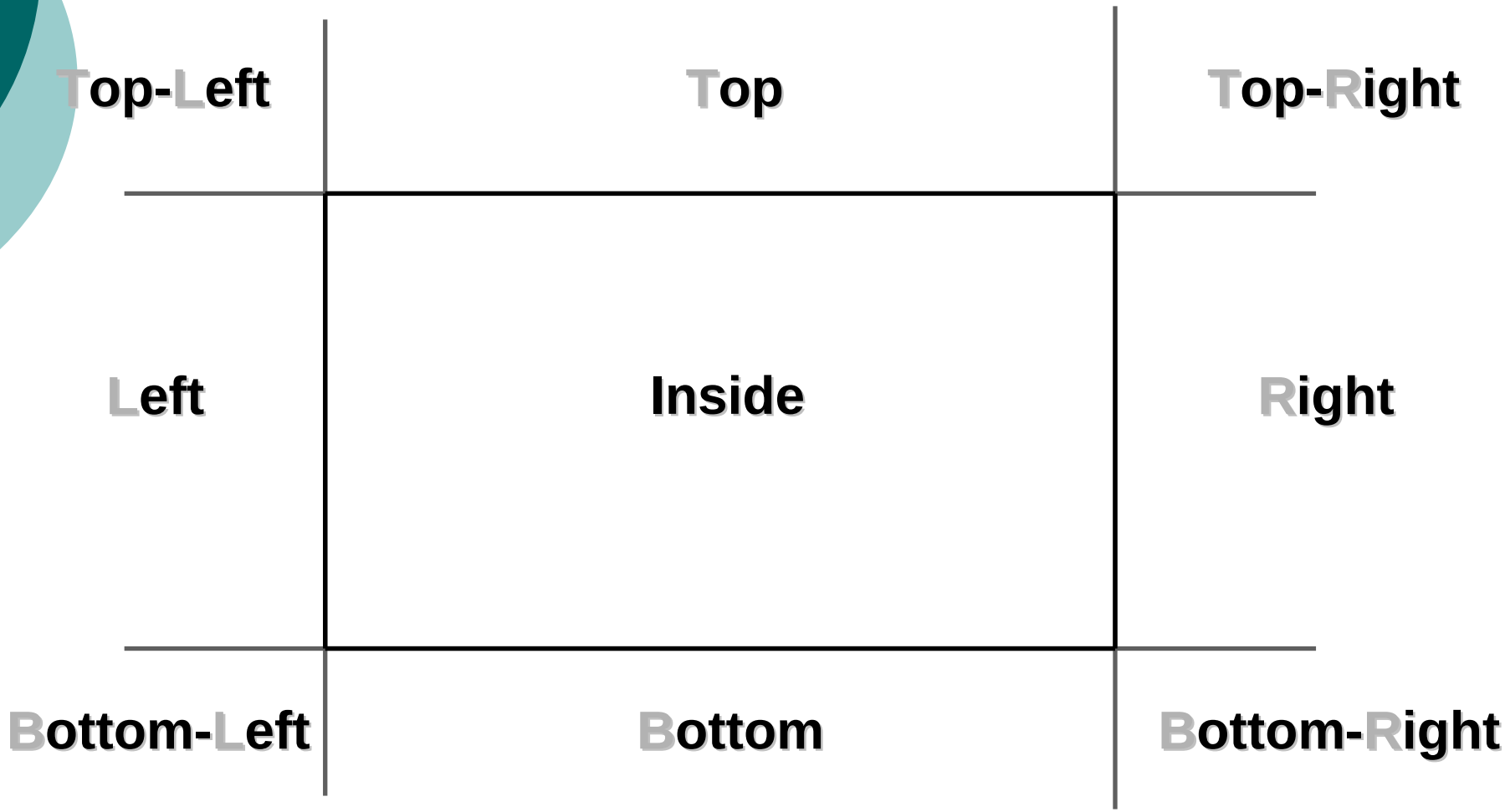
$$x = x_0 + u(x_{end} - x_0)$$

$$y = y_0 + u(y_{end} - y_0) \quad 0 \le u \le 1$$

Cohen-Sutherland Algorithm

- Oldest and most popular line-clipping algorithms.
- Method speeds up the processing of line segments by performing initial tests
- Reduces the no of intersections that must be calculated.
- Each line endpoint in picture is assigned a four digit binary code called region code .
- Region code identifies the location of the point relative to the boundaries of the clip window.

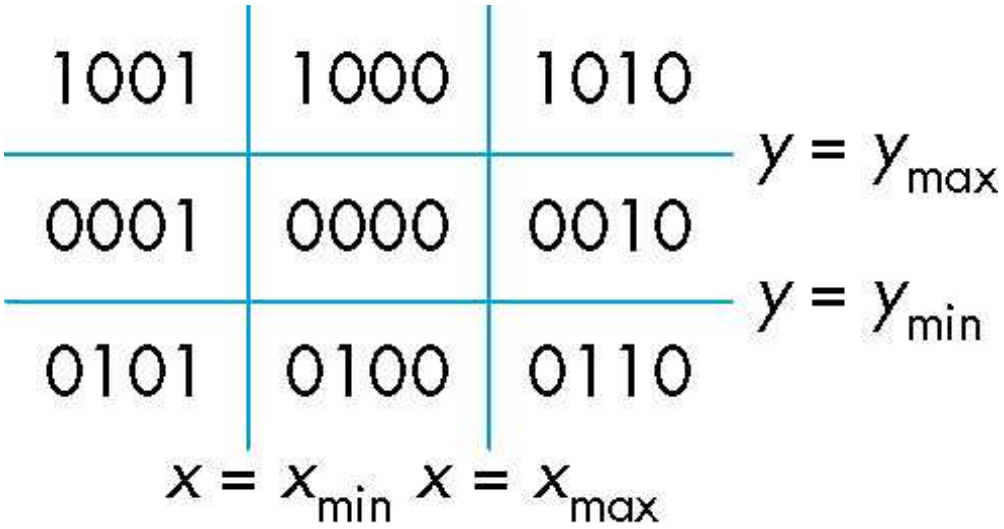
Cohen-Sutherland Algorithm



Cohen-Sutherland Algorithm (cont.)

Region Codes

Bit	1	2	3	4
	T	B	R	L



Each region is represented by a 4-bit **outcode** TBRL:

$$T = \begin{cases} 1 \text{ if } y > y_{max} \\ 0 \text{ otherwise} \end{cases} \quad B = \begin{cases} 1 \text{ if } y < y_{min} \\ 0 \text{ otherwise} \end{cases} \quad R = \begin{cases} 1 \text{ if } x > x_{max} \\ 0 \text{ otherwise} \end{cases} \quad L = \begin{cases} 1 \text{ if } x < x_{min} \\ 0 \text{ otherwise} \end{cases}$$

Trivial accept cases

- The trivial accept case corresponds to ensuring that no outcode bits are set for both endpoints. This can be nicely accomplished with a bitwise OR operation.

- Out (A) 0101
  Out (B) 0100
  bitwise OR 0101 → fails trivial accept

- Out(C) 0000
  Out (D) 0000 → trivial accept

Trivial reject cases

- The trivial reject case can be nicely accomplished with a bitwise AND operation. We can trivially reject a line segment which has a result not equal to 0000

- Out (A) 0101
  Out (B) 0100
  bitwise AND 0100 → trivial reject
- Out (E) 1000
  Out (F) 0010
  bitwise AND 0000 →fails trivial reject

Cohen-Sutherland Algorithm (cont.)

- Lines that cannot be identified as completely inside or outside by above tests can be test as follows
  - Choose an endpoint of the line that is outside the window.
  - Find the intersection point at the window boundary (base on region code).
  - Replace endpoint with the intersection point and update the region code.
  - Repeat steps until we find a clipped line either trivially accepted or trivially rejected.
- Repeat step for other lines.

How to check for intersection?

if bit 1 = 1 → there is intersection on TOP boundary.

if bit 2 = 1 → .. .. BOTTOM ..

if bit 3 = 1 → .. .. RIGHT ..

if bit 4 = 1 → .. .. LEFT ..

How to find intersection point?

- using slope intercept of the line equation

m=(p2.y-p1.y)/(p2.x -p1.x)

intersection with LEFT or RIGHT boundary.

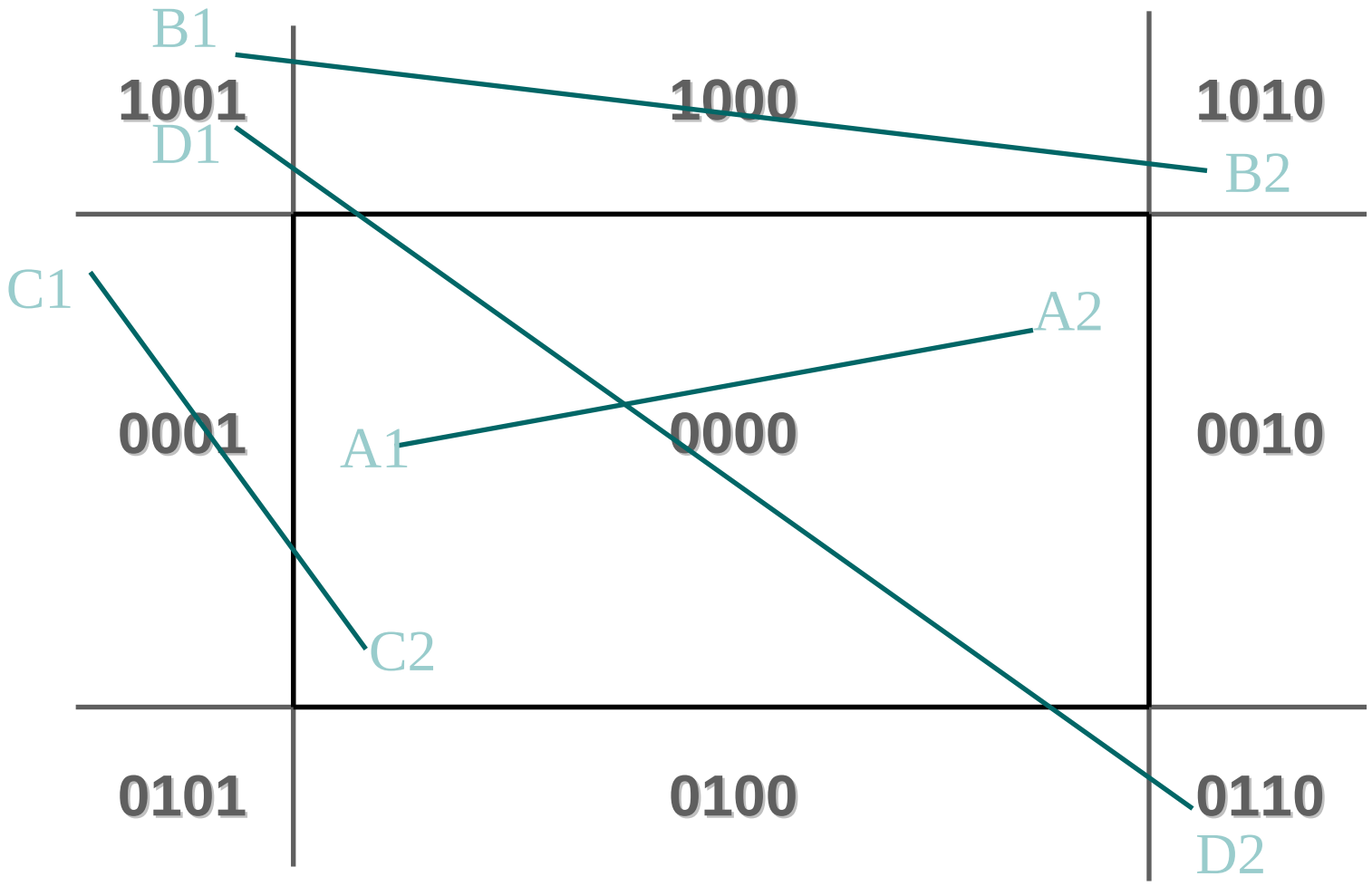
$x = xw_{min}$ (LEFT) $x = xw_{max}$ (RIGHT)

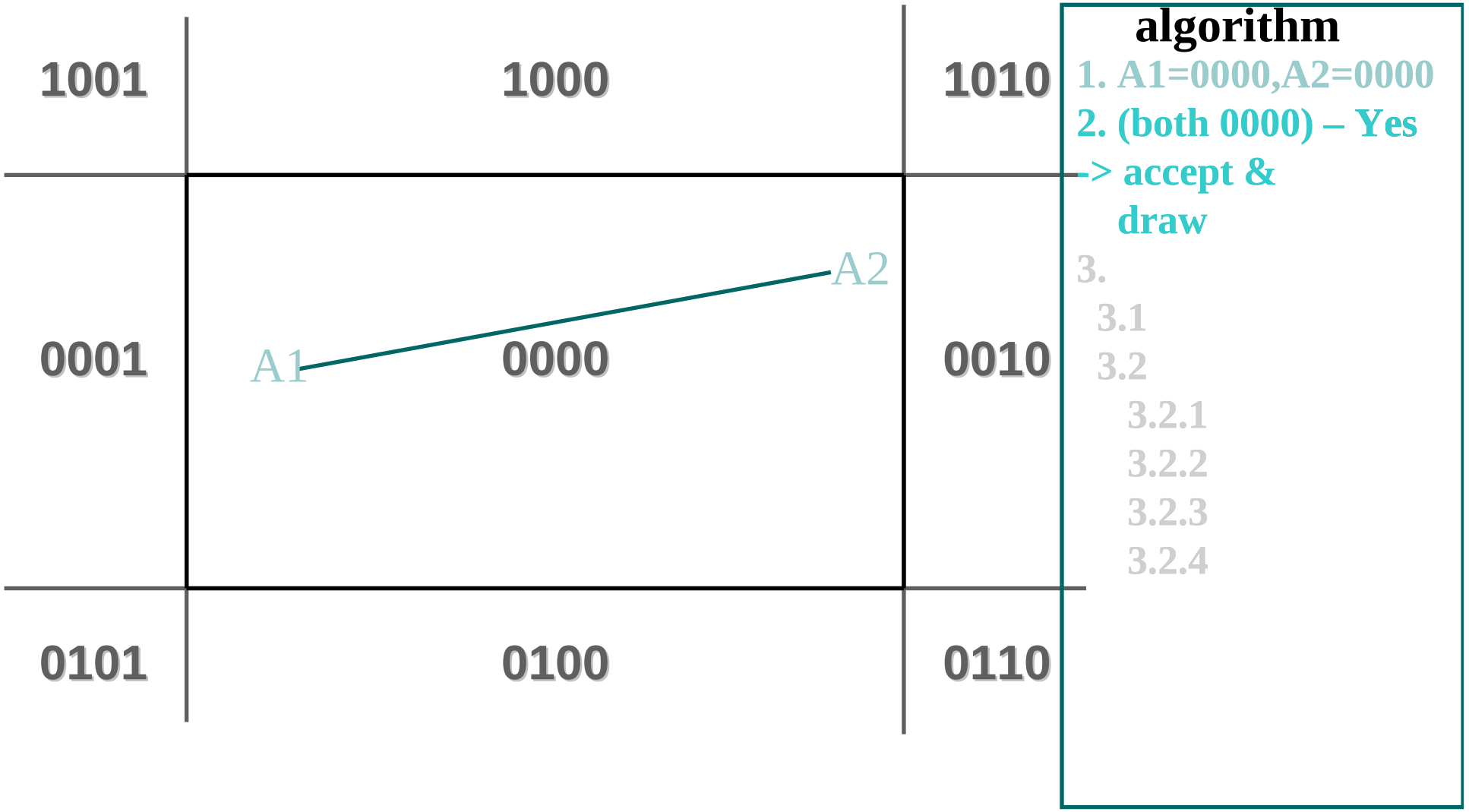
p1.y = **p1.y** + m(x –**p1.x)**

intersection with BOTTOM or TOP boundary.

$y = yw_{min}$ (BOTTOM) $y = yw_{max}$ (TOP)

p1.x = **p1.x** + (y –**p1.x**)/m

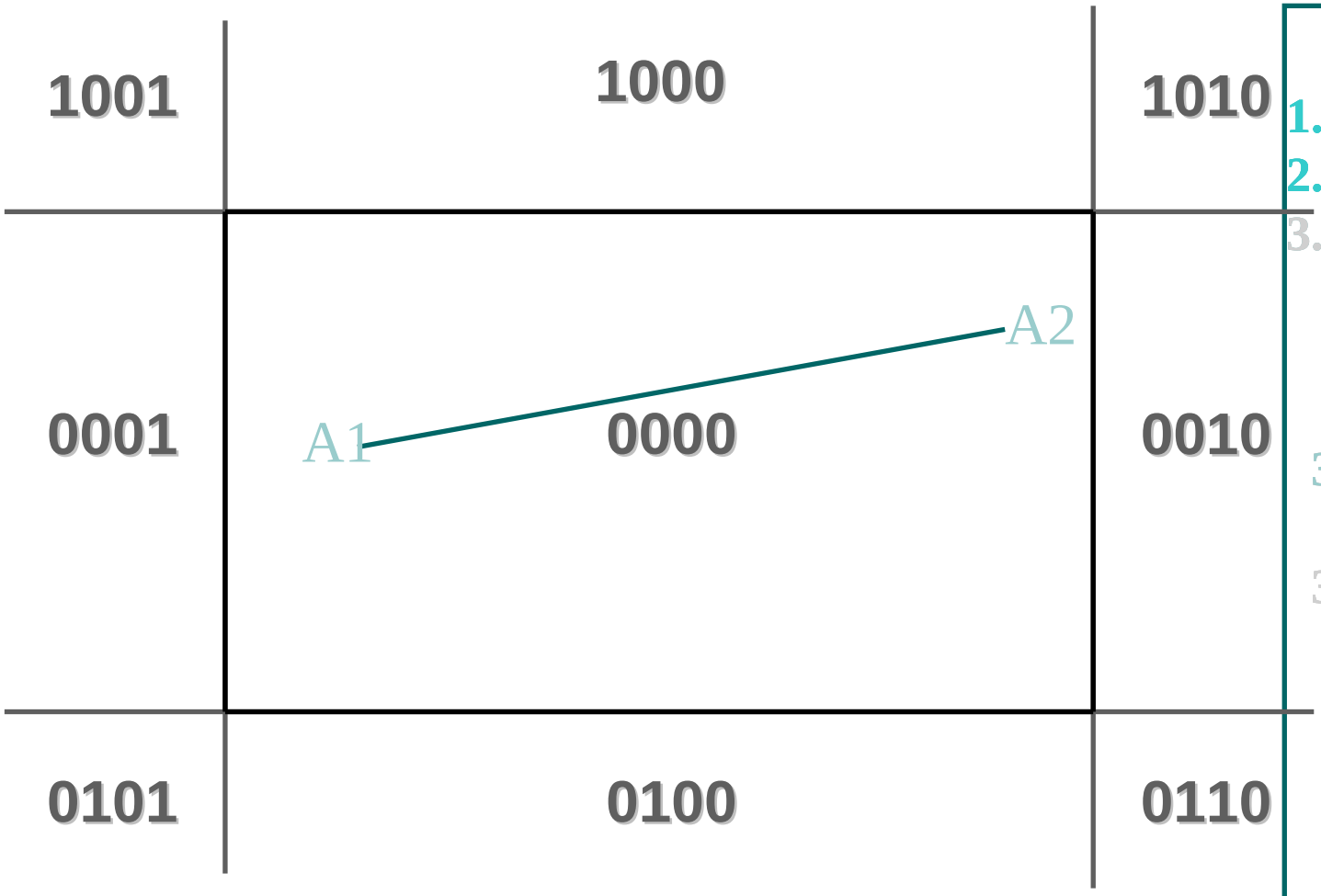

B1
1001
1000
1010
D1
B2
C1
A2
0001
A1
0000
0010
C2
0101
0100
0110
D2



algorithm

1. A1=0000,A2=0000
2. (both 0000) – Yes
-> accept &
draw
3.
   3.1
   3.2
      3.2.1
      3.2.2
      3.2.3
      3.2.4

Example



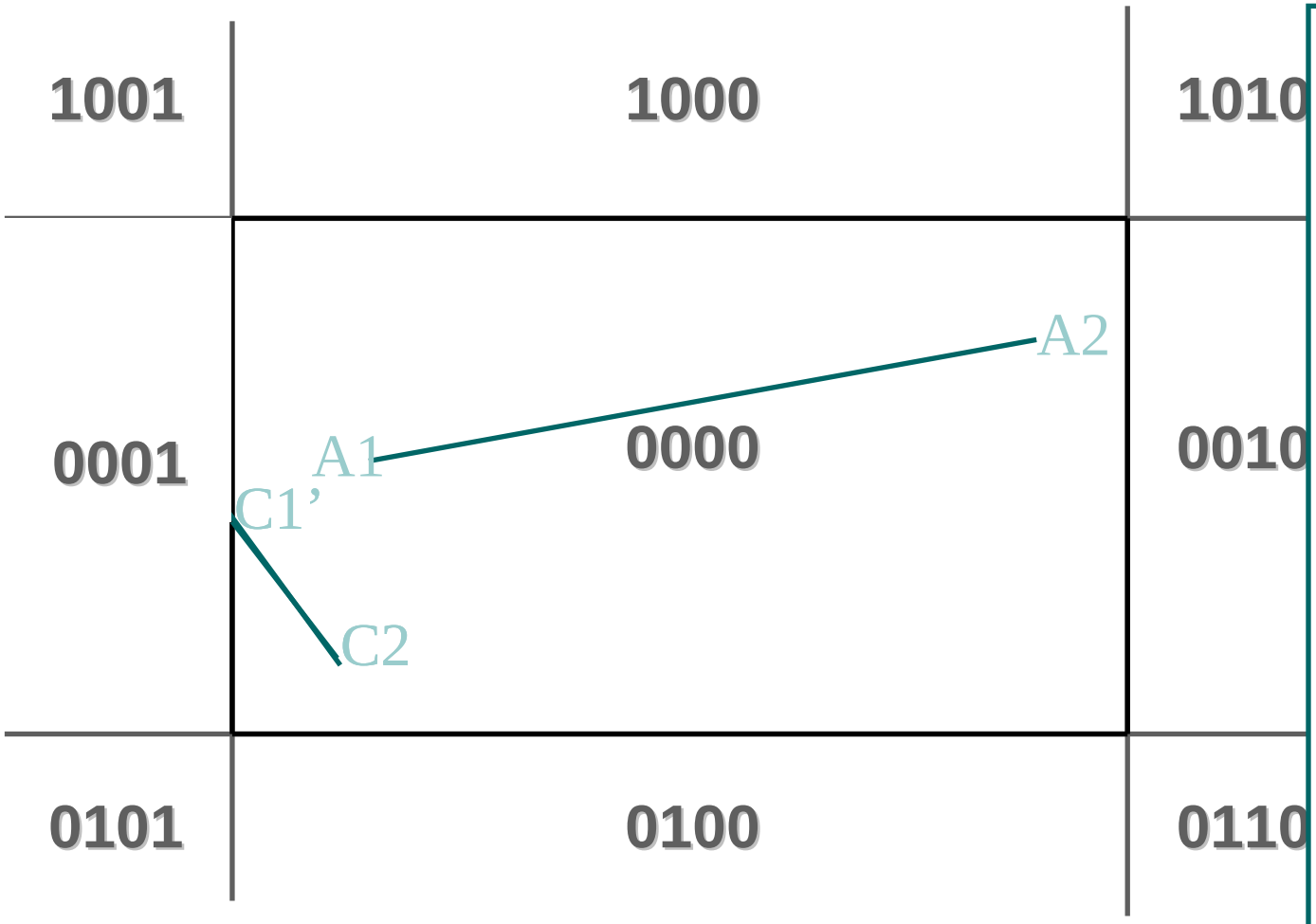
algorithm

1. B1=1001,B2=1010
2. (both 0000) – No
3. AND Operation
   - B1 → 1001
   - B2 → 1010
   - Result 1000

3.1 (not 0000) – Yes
→ reject

3.2
- 3.2.1
- 3.2.2
- 3.2.3
- 3.2.4

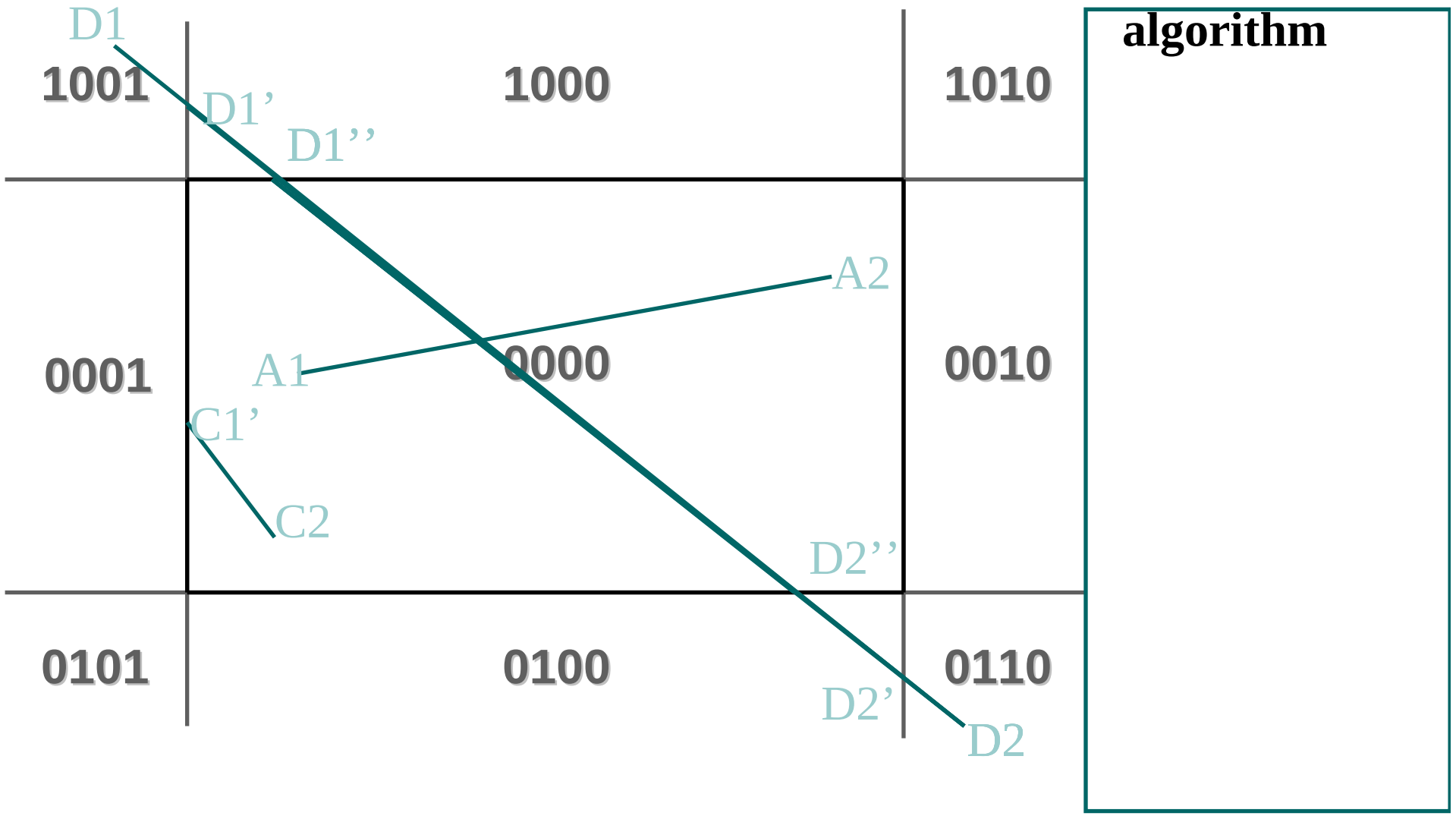
Example



algorithm

1. C1=0001,C2=0000
2. (both 0000) – Yes
-> accept &
draw
3.
   3.1
   3.2
      3.2.1
      3.2.2
      3.2.3
      3.2.4

Example


D1
1001
1000
1010
D1’
D1’’
A2
0001
A1
0000
0010
C1’
C2
D2’’
0101
0100
0110
D2’
D2
algorithm

Advantage & Disadvantage of Cohen-Sutherland Algorithm

- Easy to implement
- Early accept/reject tests
- Slow for many clipped lines

Exercise

- $(xw_{min}, xw_{max}) = (10, 150)$
- $(yw_{min}, yw_{max}) = (10, 100)$
- $P_0 = (0, 120)$
- $P_1 = (130, 5)$

Liang-Barskyn Line Clipping

- This algorithm uses the parametric equations for a line
- Solves four inequalities to find the range of the parameter for which the line is in the viewport (window).
- Consider the parametric definition of a line:
  - $x = x_1 + u\Delta x$
  - $y = y_1 + u\Delta y$
  - $\Delta x = (x_2 - x_1)$, $\Delta y = (y_2 - y_1)$, $0 \leq u \leq 1$

Liang-Barsky Line Clipping

- Mathematically, this can be written using point clipping conditions in the parametric form
  - $x_{min} \leq x_1 + u\Delta x \leq x_{max}$
  - $y_{min} \leq y_1 + u\Delta y \leq y_{max}$
- Rearranging, we get
  - $-u\Delta x \leq (x_1 - x_{min})$
  - $u\Delta x \leq (x_{max} - x_1)$
  - $-u\Delta y \leq (y_1 - y_{min})$
  - $u\Delta y \leq (y_{max} - y_1)$

Liang-Barsky Line Clipping

In general: u * pk ≤ qk

Where the parameters p and q are defined as

$$\overline{P}_1 = -\Delta x, \quad \overline{q}_1 = x_1 - x_{\min}$$

$$\overline{P}_2 = \Delta x, \quad \overline{q}_2 = x_{\max} - x_1$$

$$\overline{P}_3 = -\Delta y, \quad \overline{q}_3 = y_1 - y_{\min}$$

$$\overline{P}_4 = \Delta y, \quad \overline{q}_4 = y_{\max} - y_1$$

Liang-Barsky Line Clipping

Cases: pk = 0

- Line is parallel to boundaries
- If for the same k, qk < 0,
- The line completely outside ,reject

  Else,
- The line is inside the parallel clipping accept the line

For a non zero value of **pk,**

- calculate the value of u that corresponds to the point where the infinitely extended line intersects with the extension of boundary k as

$$u = qk / pk$$

Calculate the parameters u1 and u2 the part of the line within clip rectangle

Liang-Barsky Line Clipping

Initialize u1=0,u2=1

Case **pk < 0**

Line proceeds from outside to inside of the infinite extension

- rk = qk / pk
- u1 = max(rk, u1)

- Case **p_k > 0**

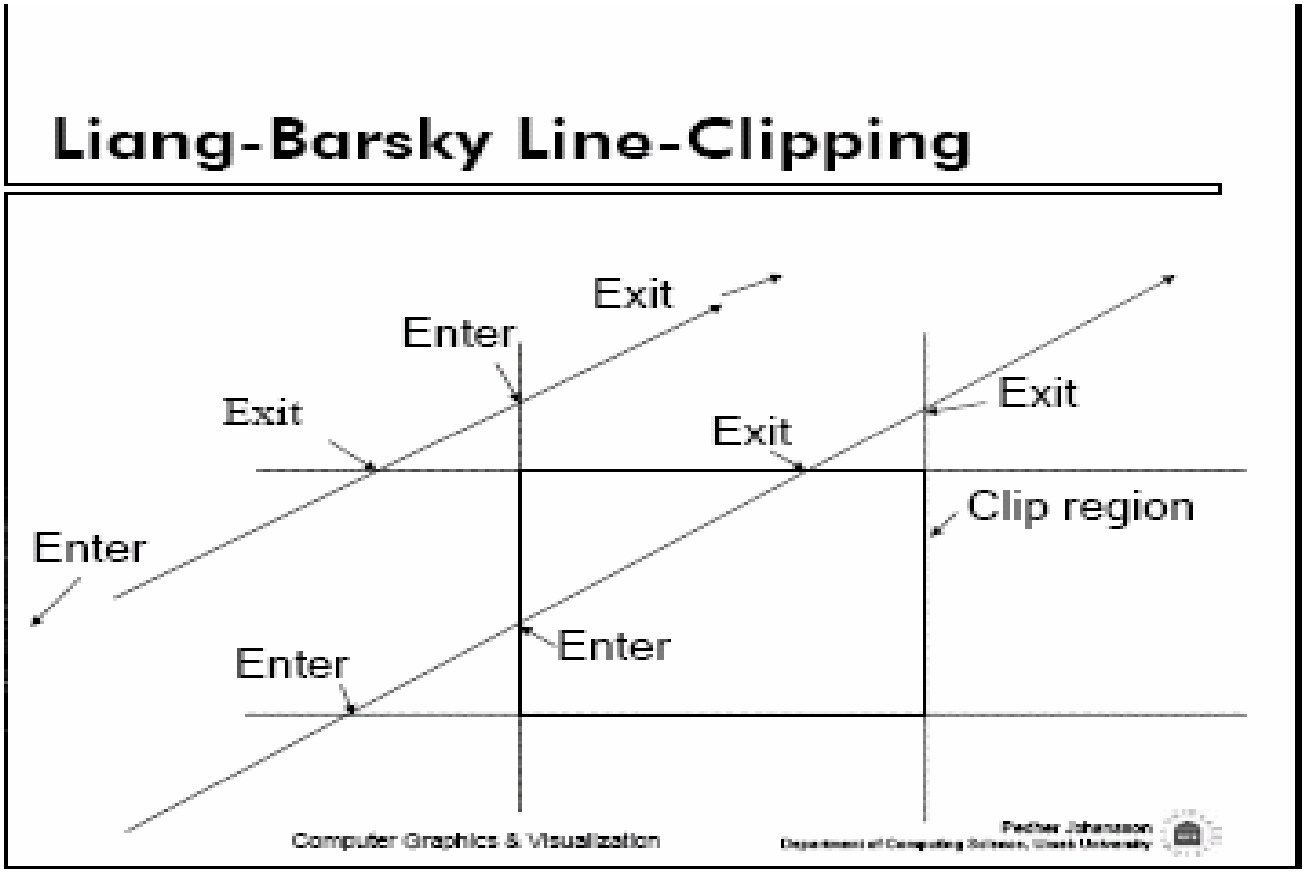
  Line proceeds from inside to outside boundary

  $r_k = qk / pk$

  $u_2 = \min(r_k, u_2)$

- If $u_1 > u_2$, the line is completely outside and can rejected
- Else
- The endpoints of the clipped line is calculated from the two values of parameters u
- If umin < umax then draw a line from:
- (x0 + Δx · umin, y0 + Δy · umin) to
- (x0 + Δx · umax, y0 + Δy · umax)

Liang-Barsky Line Clipping



Liang-Barsky Line Clipping

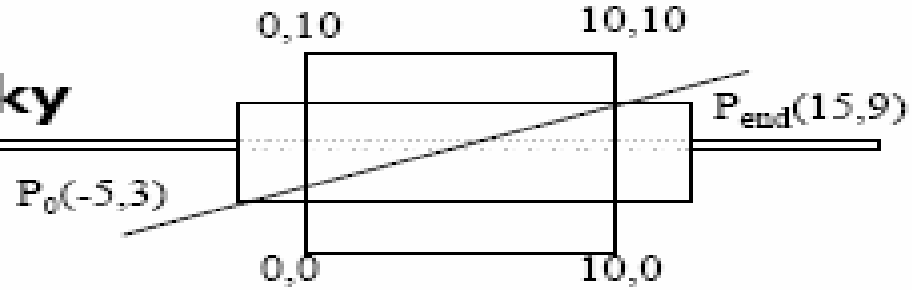
- Also extends to 3D
  - Just add equations for $z = z_1 + u\Delta z$
    - 2 more p's and q's

Liang-Barsky Line Clipping

- In most cases, Liang-Barsky is slightly more efficient
  - Intersection calculations are reduced
  - Avoids multiple shortenings of line segments
  - Window intersection of the line is computed only once

Liang-Barsky Line Clipping

Example Liang-Barsky



$$u_{left} = \frac{q_1}{p_1} = \frac{x_0 - xw_{min}}{-\Delta x} = \frac{-5-0}{-(15-(-5))} = \frac{1}{4}$$

Entering ⇒ $u_{min} = 1/4$

$$u_{right} = \frac{q_2}{p_2} = \frac{xw_{max} - x_0}{\Delta x} = \frac{10-(-5)}{15-(-5)} = \frac{3}{4}$$

Exiting ⇒ $u_{max} = 3/4$

$$u_{bottom} = \frac{q_3}{p_3} = \frac{y_0 - yw_{min}}{-\Delta y} = \frac{3-0}{-(9-3)} = -\frac{1}{2}$$

u < 0 then ignore

$$u_{top} = \frac{q_4}{p_4} = \frac{yw_{max} - y_0}{\Delta y} = \frac{10-3}{9-3} = \frac{7}{6}$$

u > 1 then ignore

Pauline Johnston
Department of Computing Science, Umeå University

Liang-Barsky Line Clipping

Liang-Barsky Line-Clipping

- We have $u_{min} = 1/4$ and $u_{max} = 3/4$

$$P_{end} - P_0 = (15+5, 9-3) = (20,6)$$
$$\Delta x \quad \Delta y$$

- If $u_{min} < u_{max}$, there is a line segment
  - compute endpoints by substituting u values
- Draw a line from
  $(-5+(20)\cdot(1/4),\ 3+(6)\cdot(1/4))$
  to
  $(-5+(20)\cdot(3/4),\ 3+(6)\cdot(3/4))$

Fredrik Johansson
Department of Computing Science, Umeå University

- Thank You