Clipping

CMSC 161: Interactive Computer Graphics

2nd Semester 2014-2015

Institute of Computer Science

University of the Philippines – Los Baños

Lecture by James Carlo Plaras

Clipping

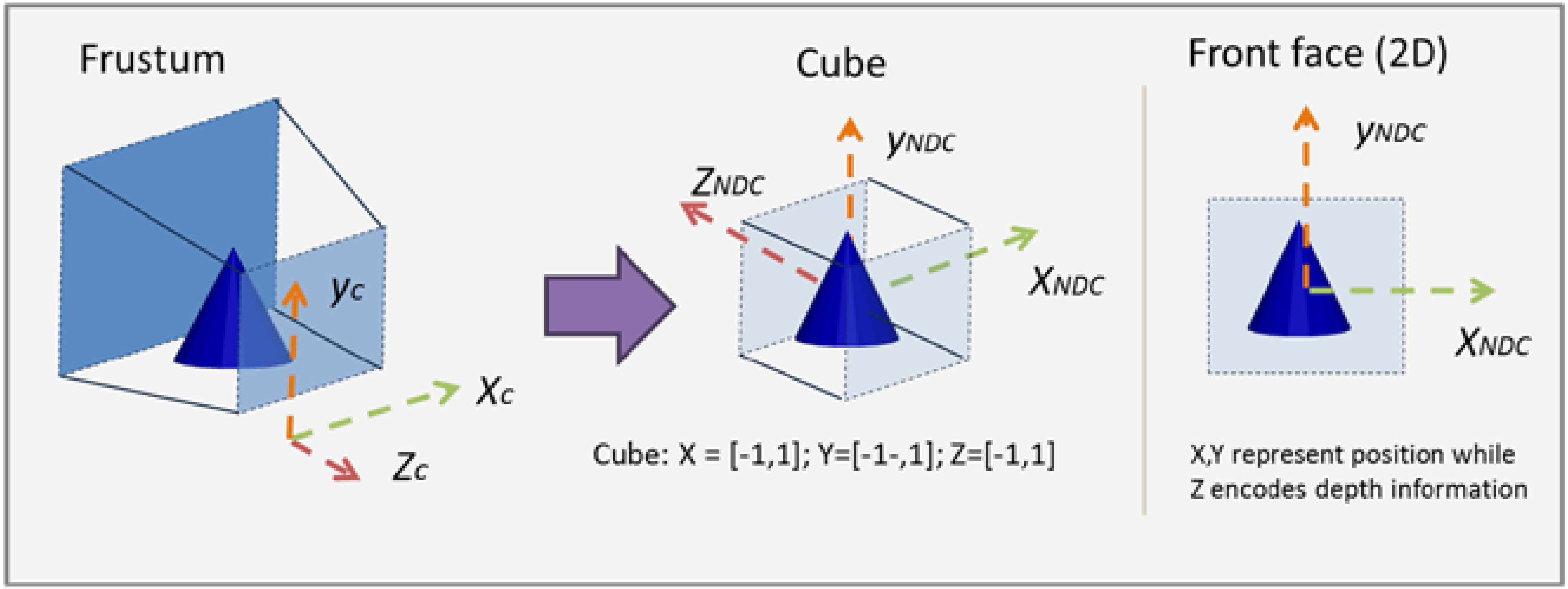
The process of determining which primitives (or its parts) fit within the view volume

Clipping

Primitives pass through the clipper are **accepted**

Primitives that cannot appear are **rejected/culled**

Normalized Device Coordinates

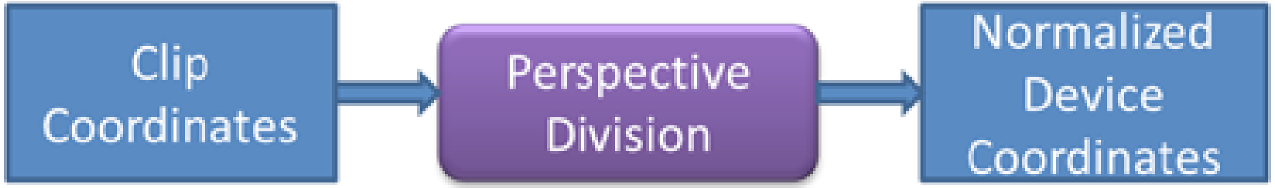

Frustum

Cube

Front face (2D)

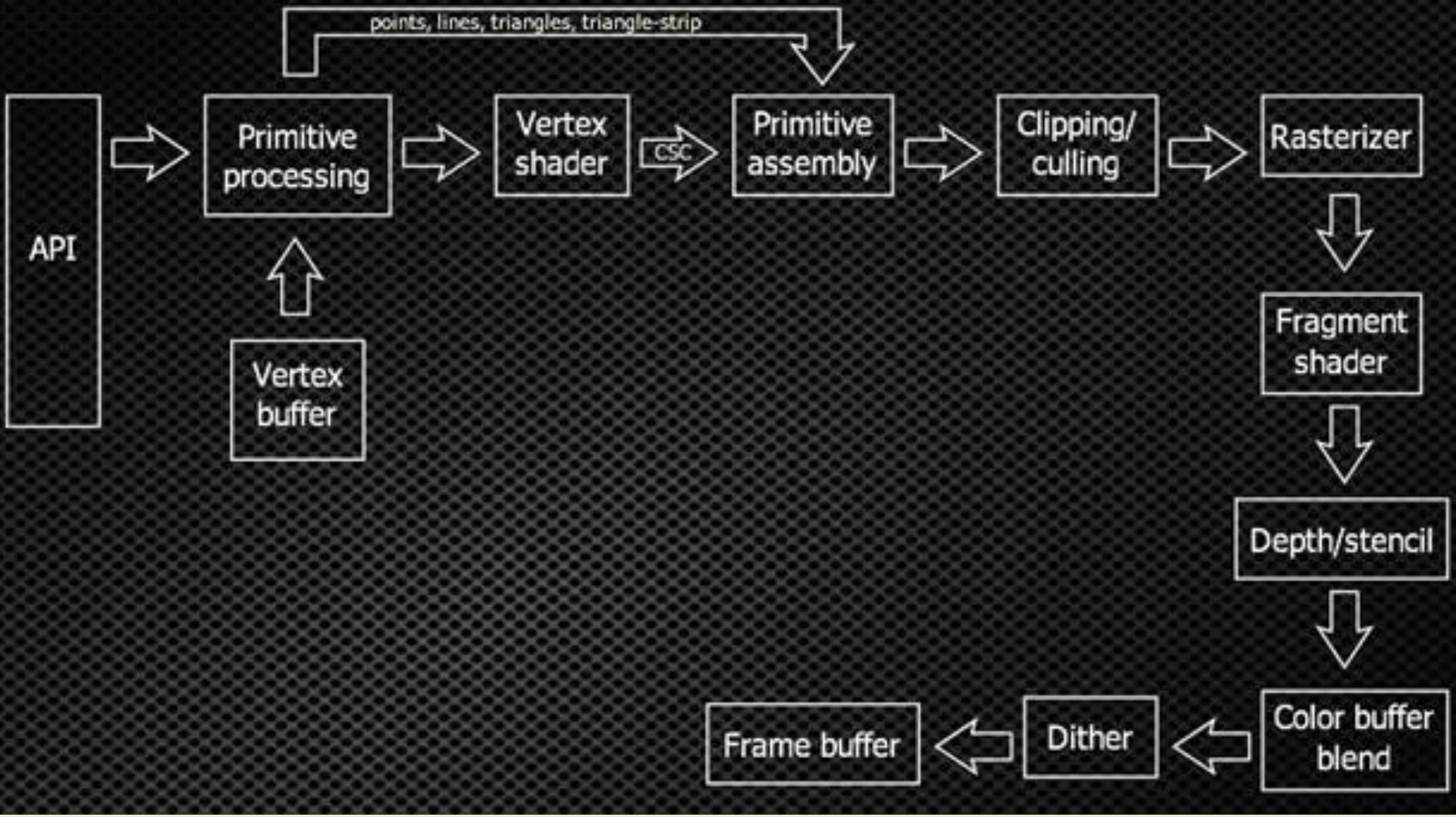
Cube: X = [-1,1]; Y=[-1-,1]; Z=[-1,1]

X,Y represent position while
Z encodes depth information

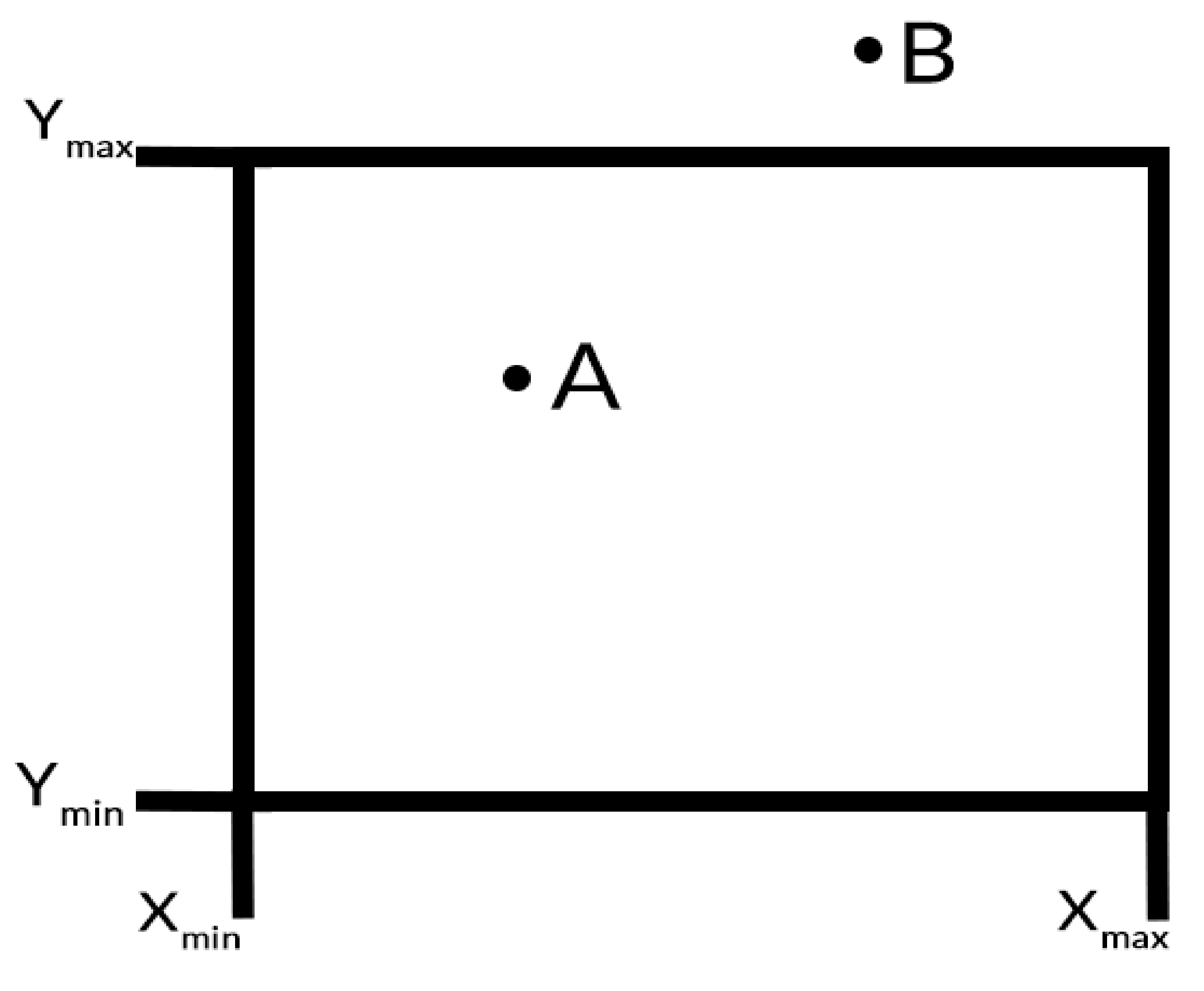

Clip Coordinates

Perspective Division

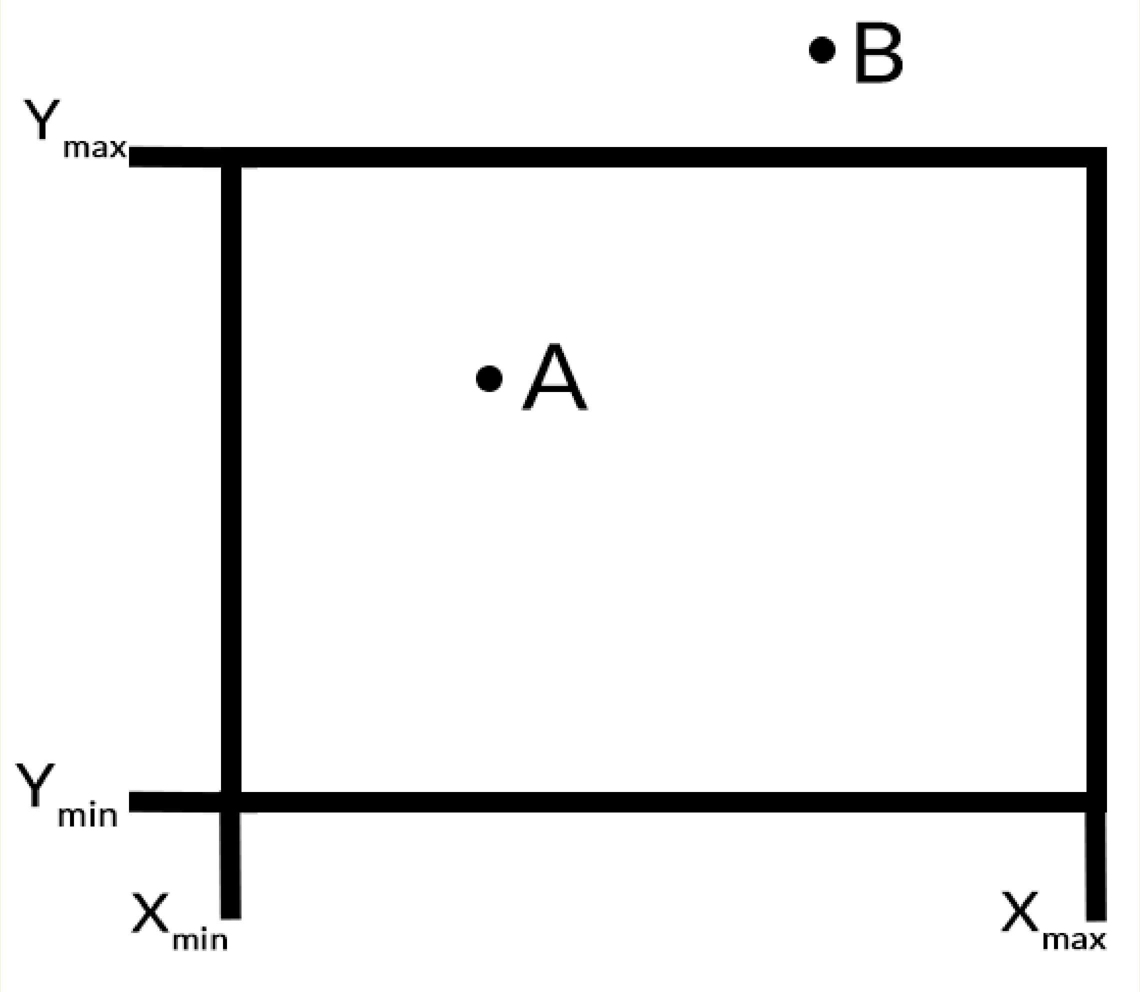
Normalized Device Coordinates


points, lines, triangles, triangle-strip
API
Primitive processing
Vertex buffer
Vertex shader
CSC
Primitive assembly
Clipping/ culling
Rasterizer
Fragment shader
Depth/stencil
Color buffer blend
Dither
Frame buffer

Point Clipping



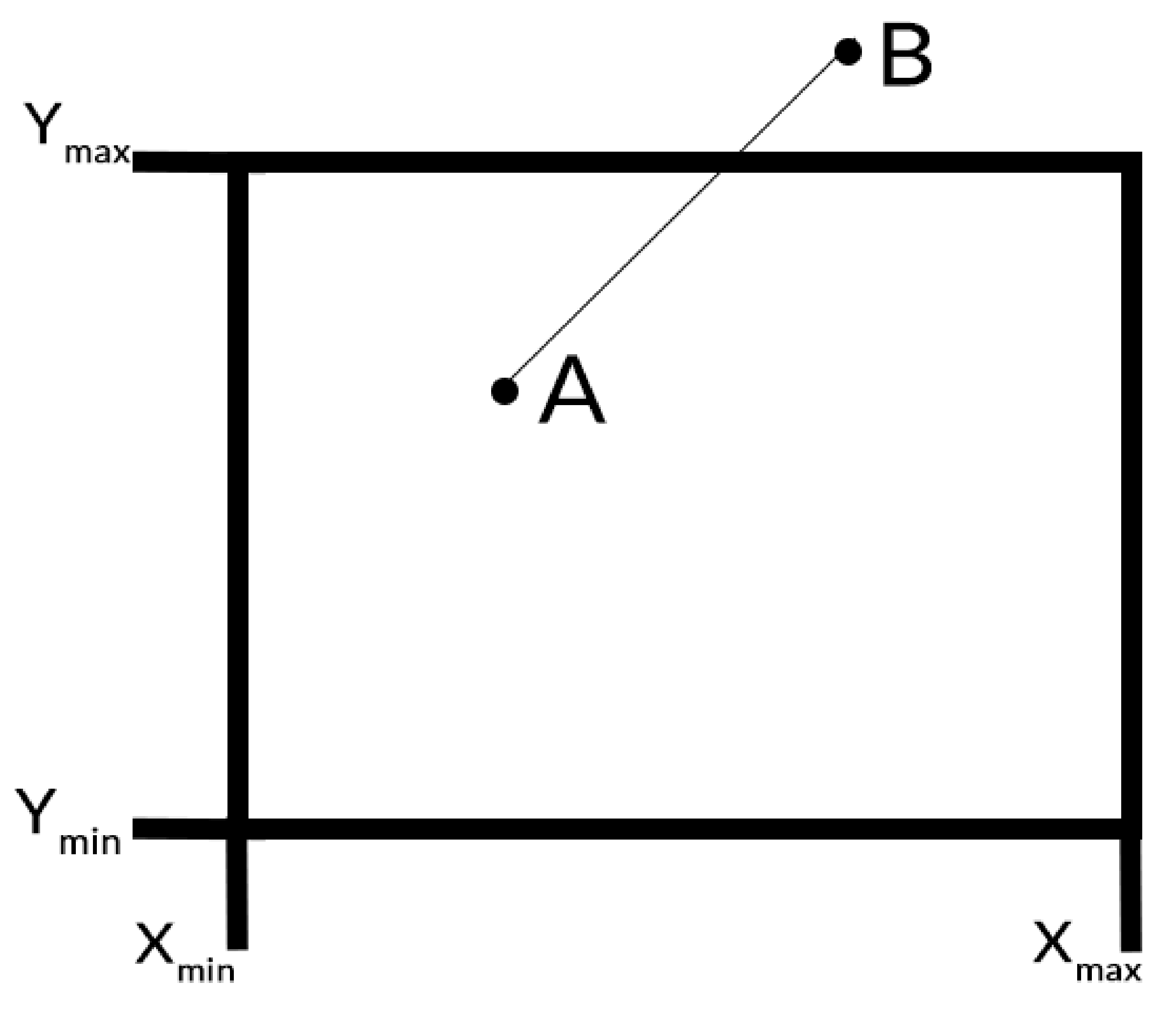
Point Clipping



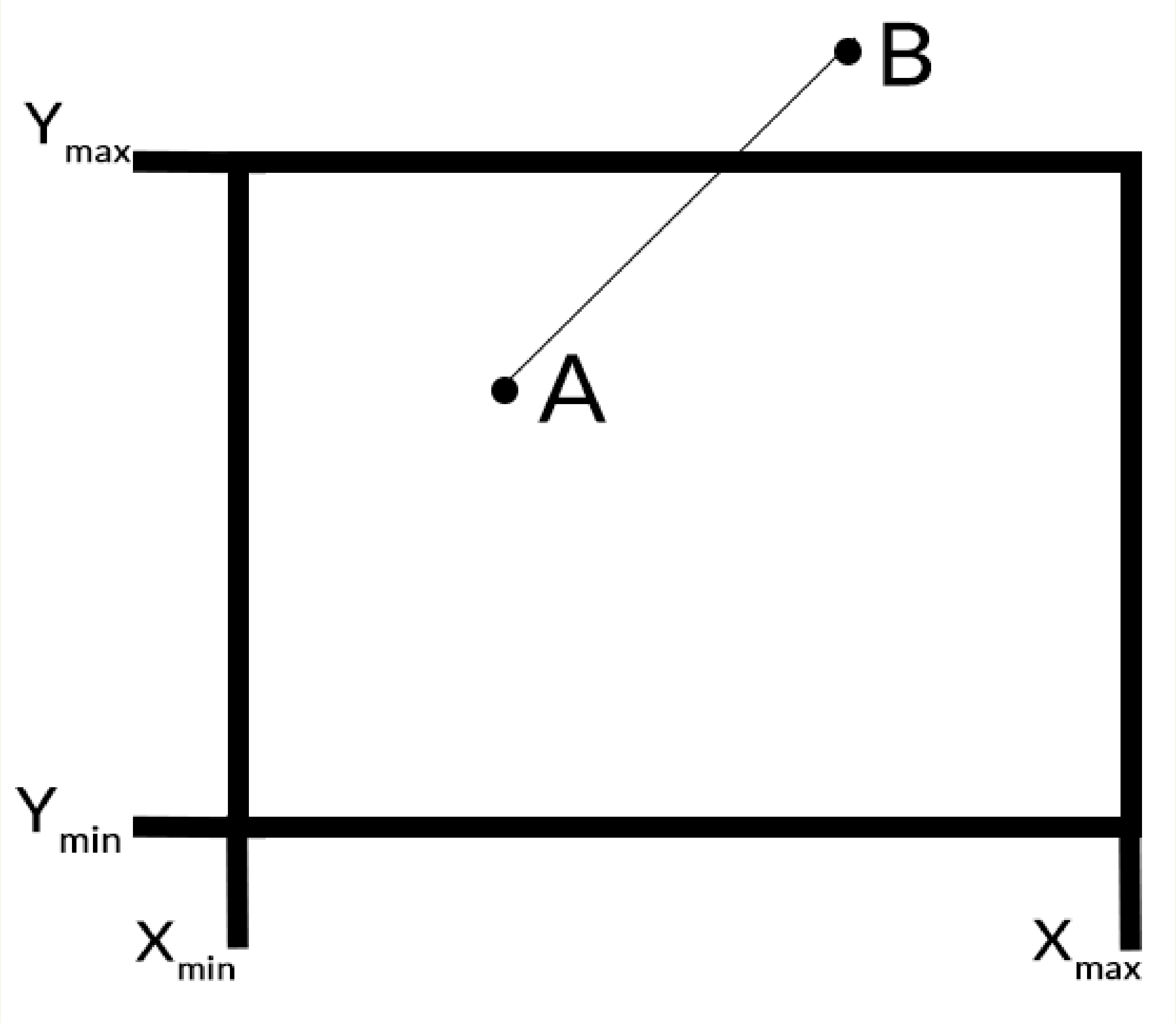
- Check if Point is inside the window

$$X_{min} \leq x \leq X_{max}$$
$$Y_{min} \leq y \leq Y_{max}$$

Line Clipping



Straight-forward Line Clipping



Compute Intersections of Line AB on lines:

$$Y = Y_{max}$$
$$Y = Y_{min}$$
$$X = X_{max}$$
$$X = X_{min}$$

Computing Line Intersection

2 points (x_1, y_1) and (x_2, y_2)

Compute equation of line

Computing Line Intersection

$$y - y_1 = m(x - x_1)$$

$$y - y_1 = \frac{(y_2 - y_1)}{(x_2 - x_1)}(x - x_1)$$

Computing Line Intersection

Ex. $(Y = Y_{max})$

Substitute the intersecting line to the equation

Computing Line Intersection

$$y - y_1 = m(x - x_1)$$

$$Y_{max} - y_1 = m(x - x_1)$$

$$\frac{Y_{max} - y_1}{m} - x_1 = x$$

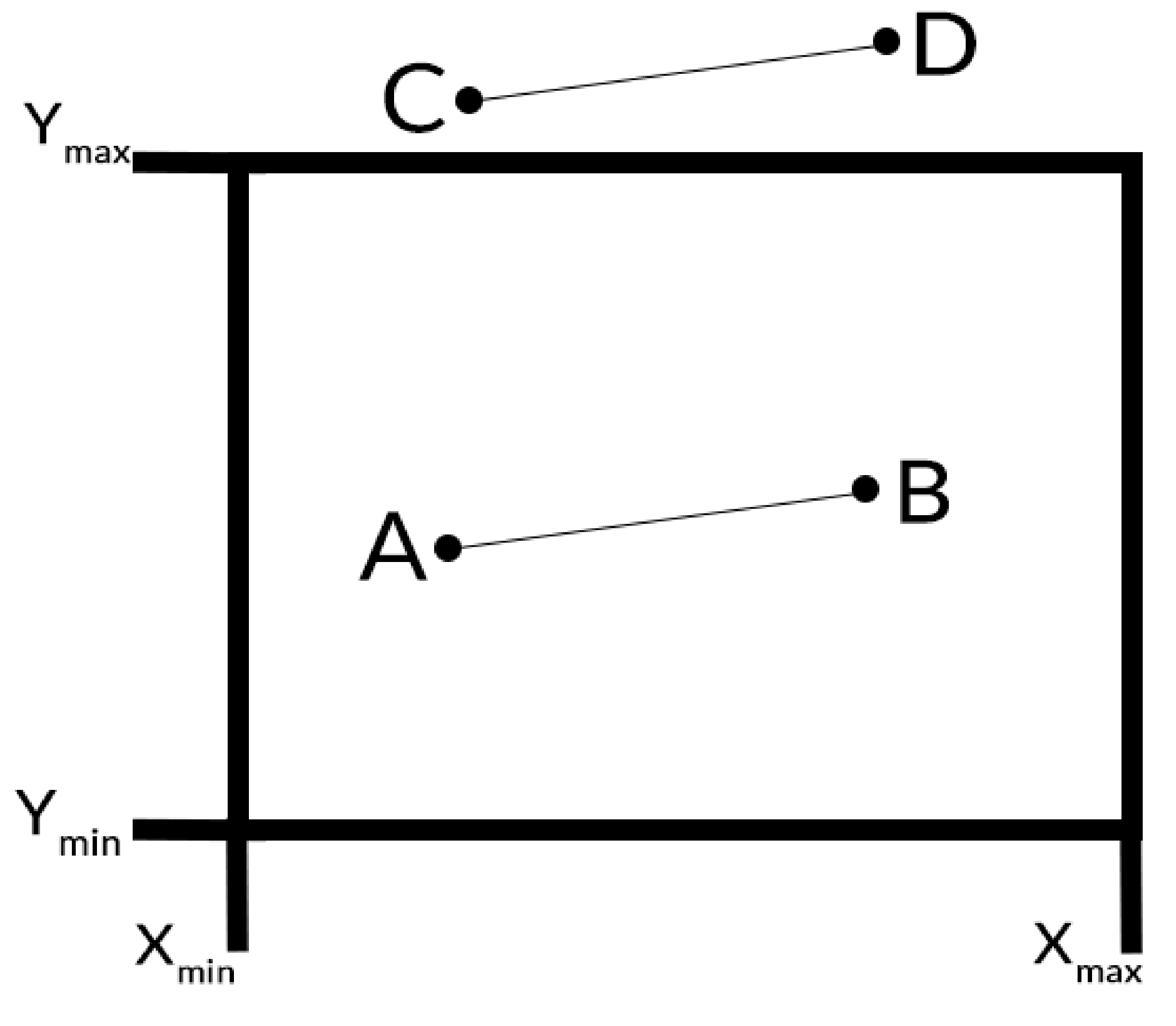
$$x = \frac{Y_{max} - y_1}{m} - x_1$$

Computing Line Intersection

New intersection point is

$$\left(\frac{Y_{max} - y_1}{m} - x_1, Y_{max}\right)$$

Straight-forward Line Clipping



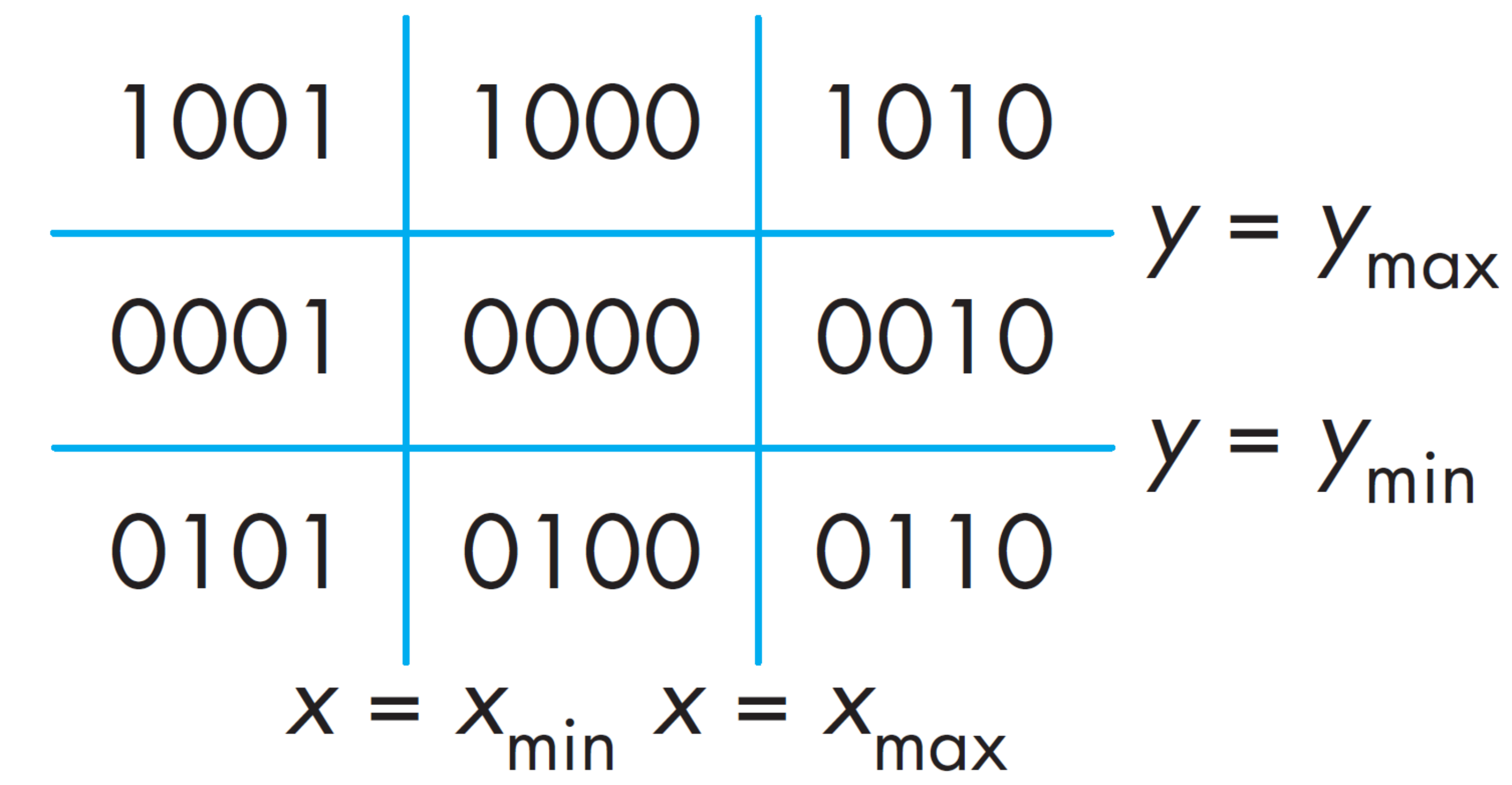
Not efficient when the lines are **trivial cases**

- Completely outside
- Completely inside

Always computes for 4 intersections

- Can only be 1 or 2 intersections

Cohen-Sutherland Algorithm



1001	1000	1010
0001	0000	0010
0101	0100	0110

Cohen-Sutherland Algorithm

Each point of a line is assigned a **outcode**

Outcode is a bit string that specifies the location on the 9 partitions

Cohen-Sutherland Algorithm

P(x,y) → [Top Bottom Right Left]

P(x,y) → [T B R L]

$$T = (Y > Y_{max})$$

$$B = (Y < Y_{min})$$

$$R = (X > X_{max})$$

$$L = (X < X_{min})$$

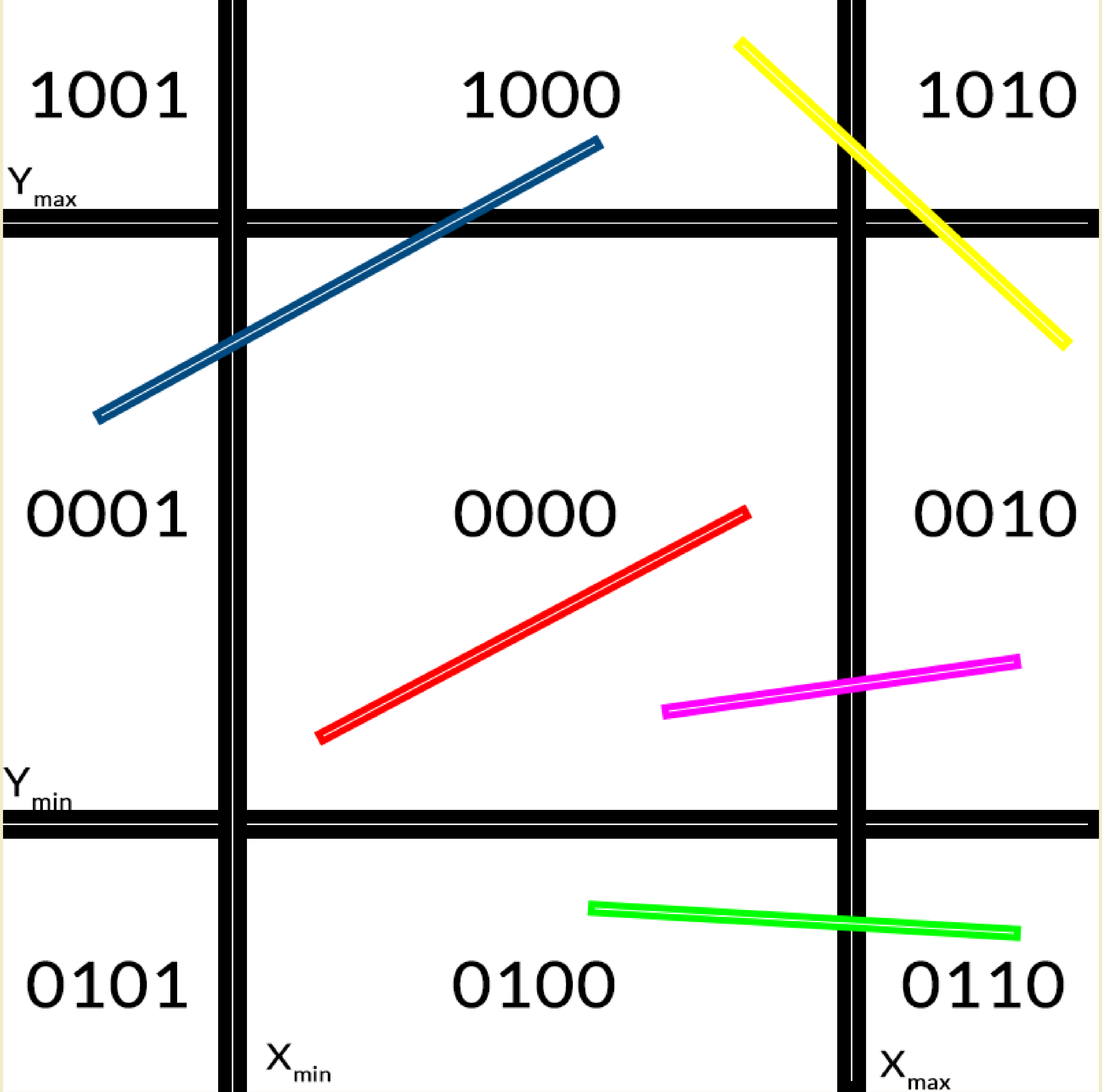
Cohen-Sutherland Algorithm

Example

$$Y_{max} = 1, Y_{min} = -1, X_{max} = 1, X_{min} = -1$$

$$P(0.5{,}1.35)$$
$$T = (1.35 > 1) = 1$$
$$B = (1.35 < -1) = 0$$
$$R = (0.5 > 1) = 0$$
$$L = (0.5 < -1) = 0$$

$$P(0.5{,}1.35) \; \rightarrow 1000$$


1001
1000
1010
Y_{max}
0001
0000
0010
Y_{min}
0101
0100
0110
X_{min}
X_{max}

Cohen-Sutherland Algorithm

Get outcodes of P_1 and P_2 of line

$$P_1 \rightarrow o_1 \text{and } P_2 \rightarrow o_2$$

Cohen-Sutherland Algorithm

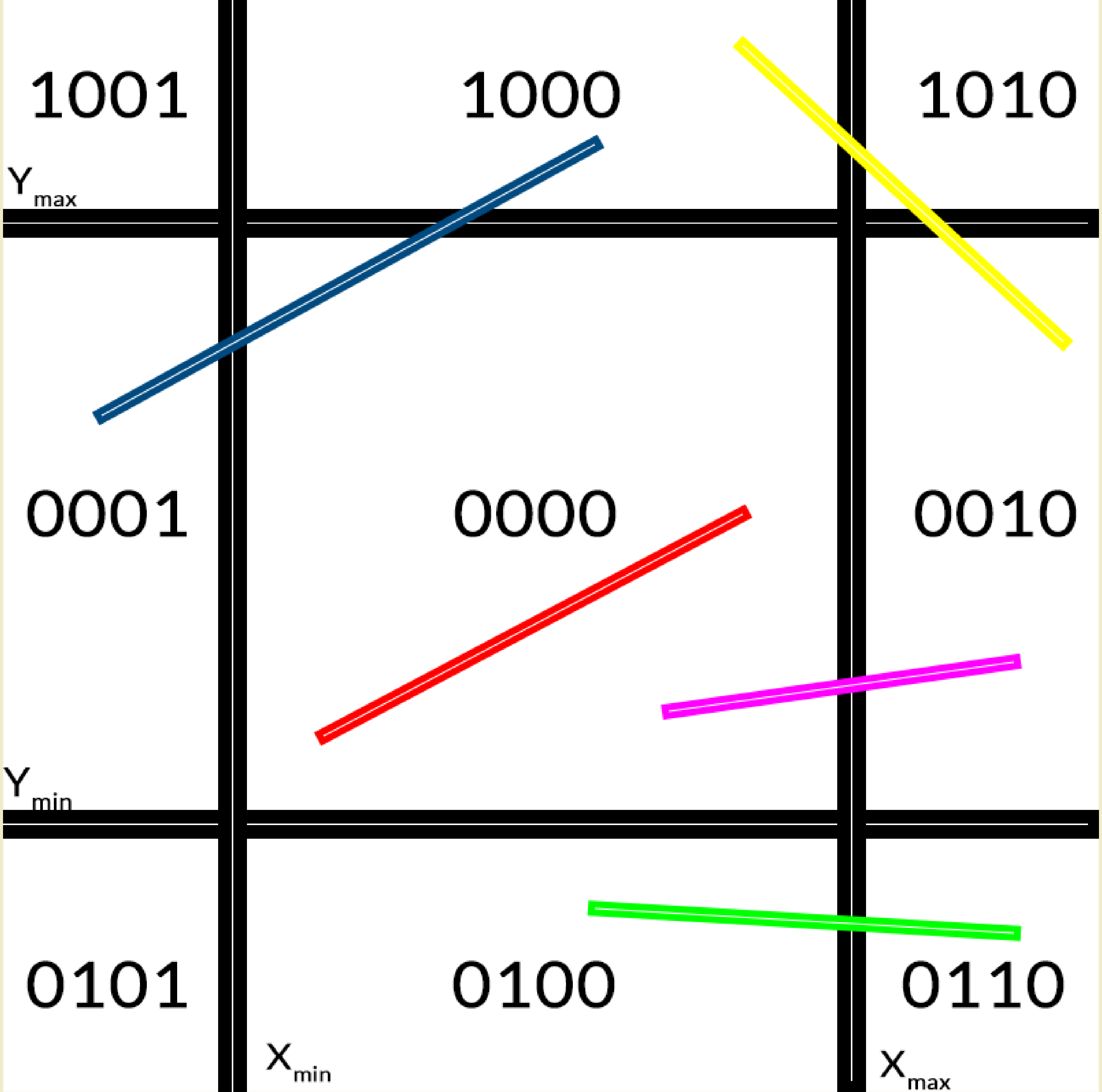
Case 1: Both outcodes are 0000 $(o_1 | o_2)$

Accept Line (Stop Testing)

Cohen-Sutherland Algorithm

Case 2: Both outside ($o_1 \& \, o_2 \; != \; 0000$)

Reject Line (Stop Testing)



1001 1000 1010

0001 0000 0010

0101 0100 0110

Cohen-Sutherland Algorithm

Case 3 - ($o_1 \& o_2 == 0000$)

Select one point with non-0000 outcode

Find the intersection point of outside point

depending on its outcode

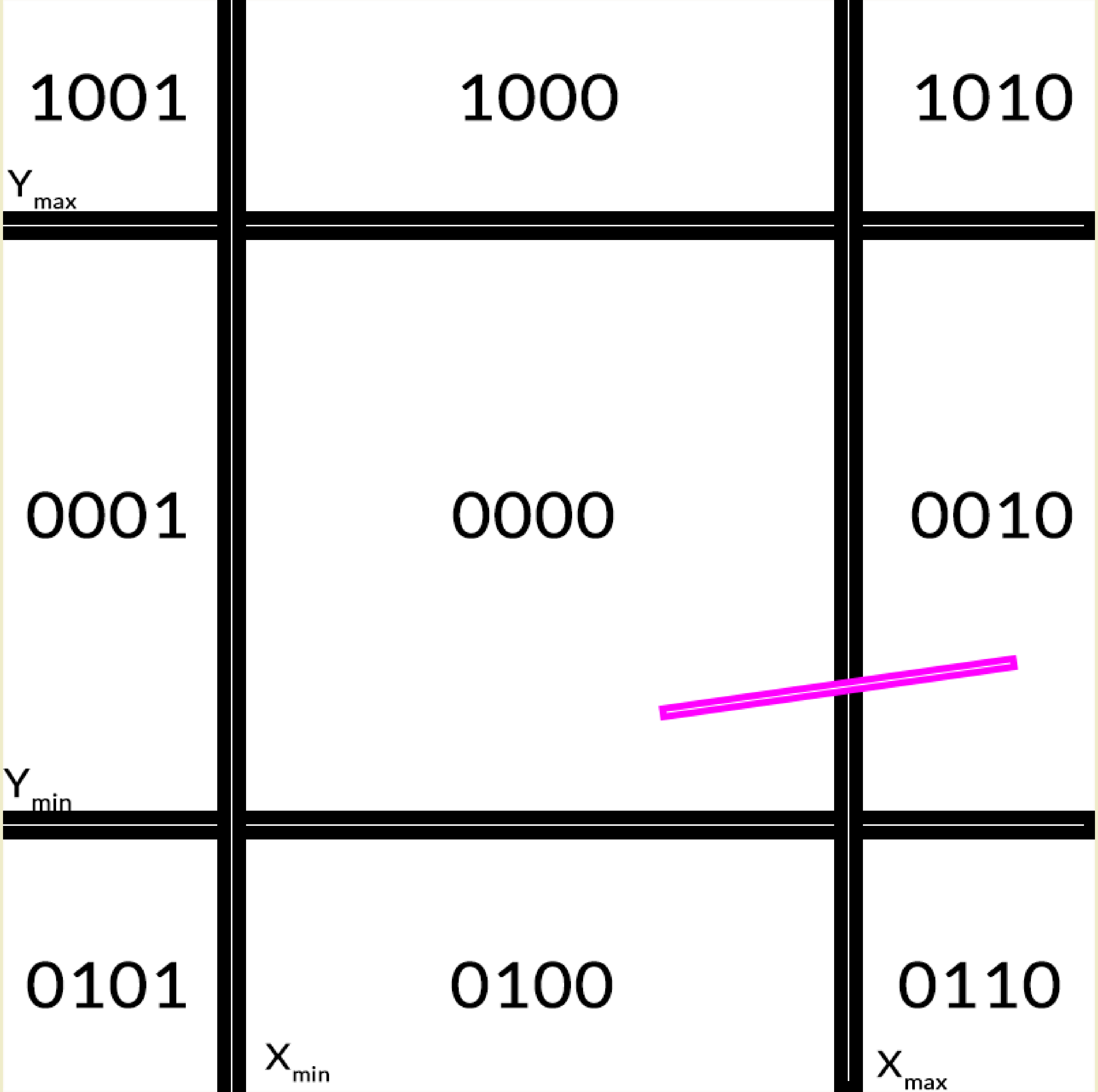
Cohen-Sutherland Algorithm

Case 3 - ($o_1 \& \, o_2 \; == \; 0000$)

The computed intersection point will replace the selected point

Update the new point’s outcode

Repeat Algorithm



1001 1000 1010

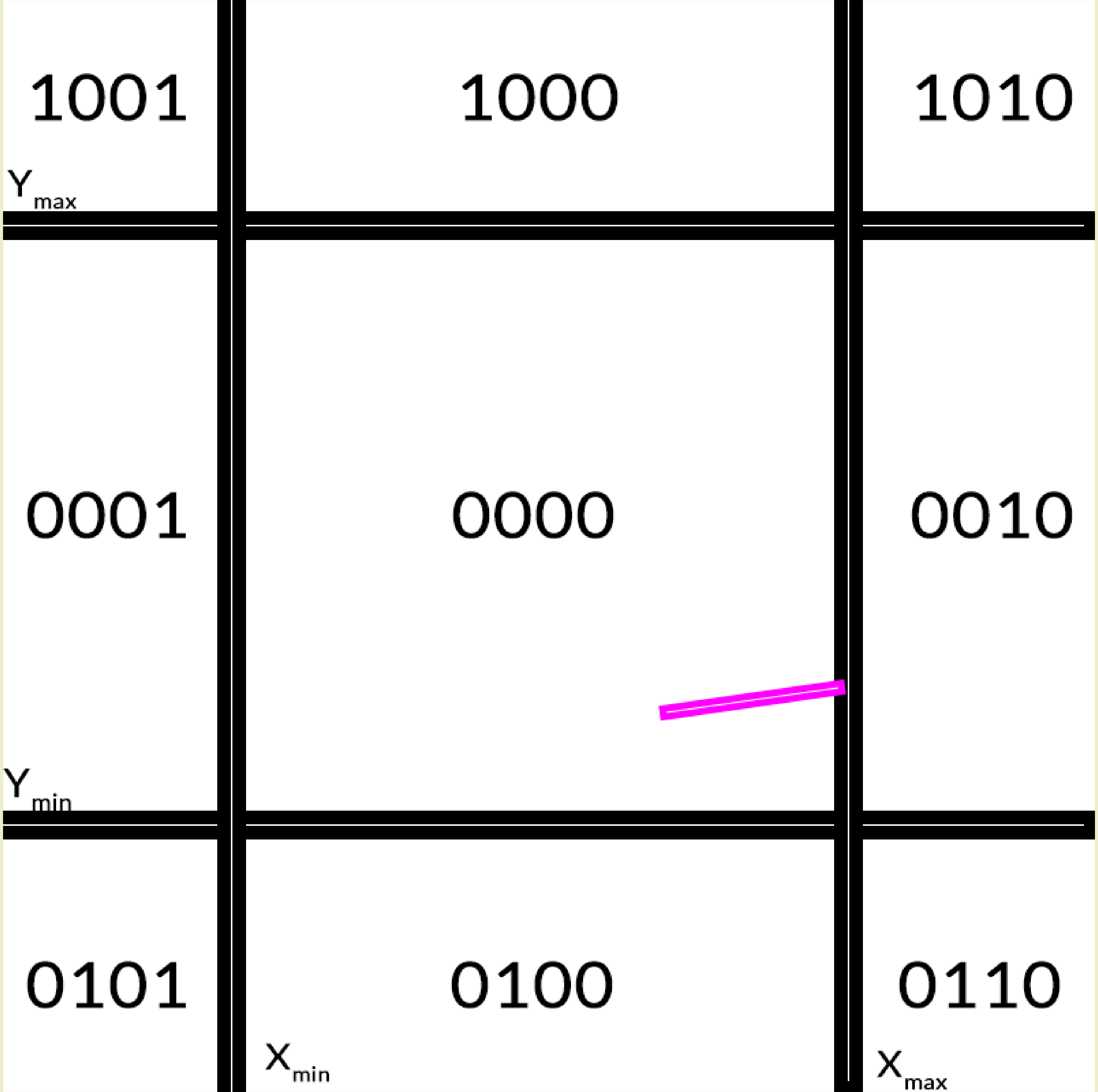
Y_{max}

0001 0000 0010

Y_{min}

0101 0100 0110

X_{min} X_{max}


1001
1000
1010
Y$_{max}$
0001
0000
0010
Y$_{min}$
0101
0100
0110
X$_{min}$
X$_{max}$

Cohen-Sutherland Algorithm

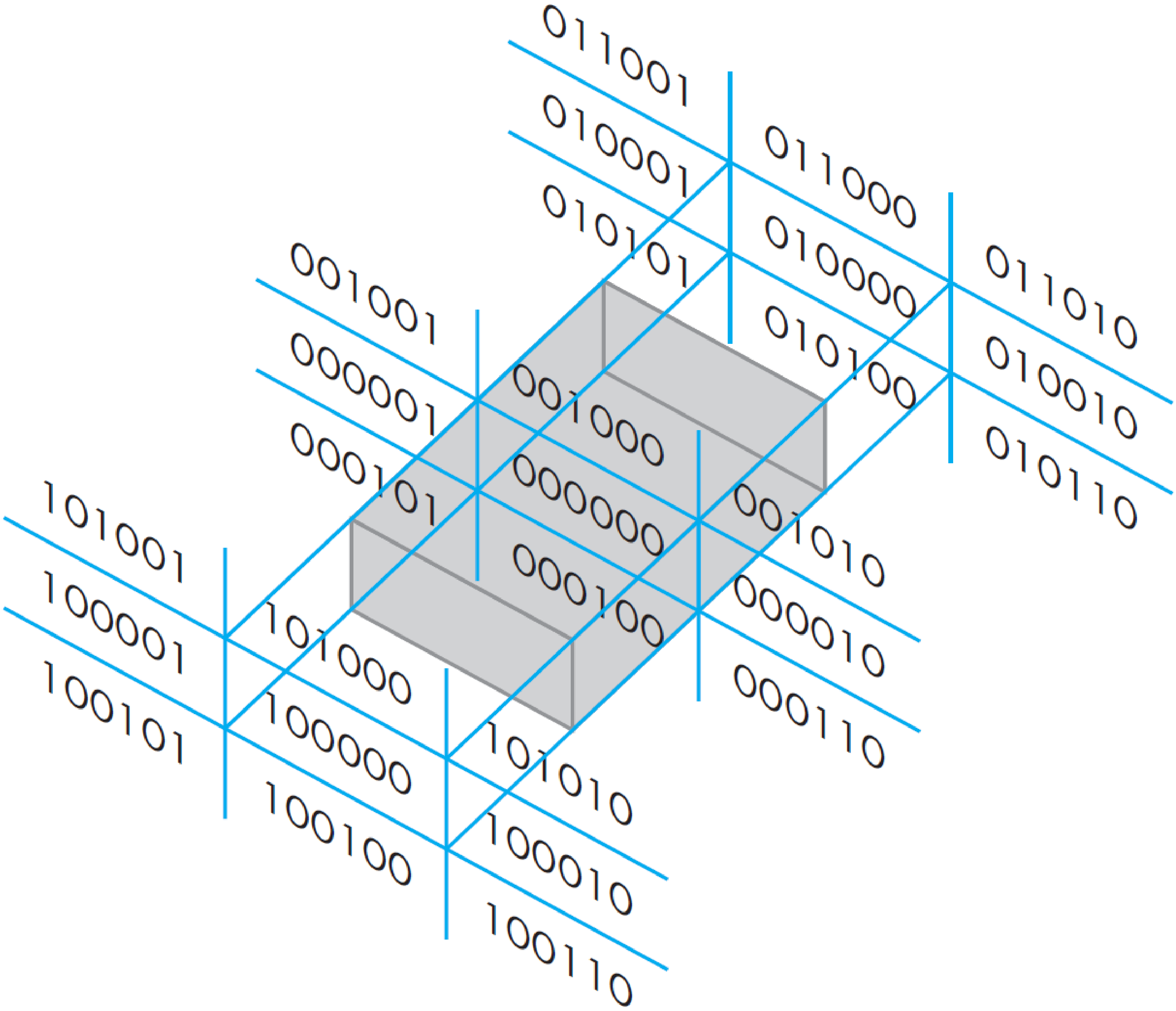
Accepts **trivial cases fast**

Calculates intersections only if necessary (based from outcodes)

Cohen-Sutherland Algorithm

Straight forward extension to 3D

27 partitions, 6 bit outcodes


011001
010001
010101
011000
010000
010100
011010
010010
010110
001001
000001
000101
001000
000000
000100
001010
000010
000110
101001
100001
100101
101000
100000
100100
101010
100010
100110

Other Line Clipping Algorithms

Cyrus-Beck algorithm

Utilizes Parametric equation of the line

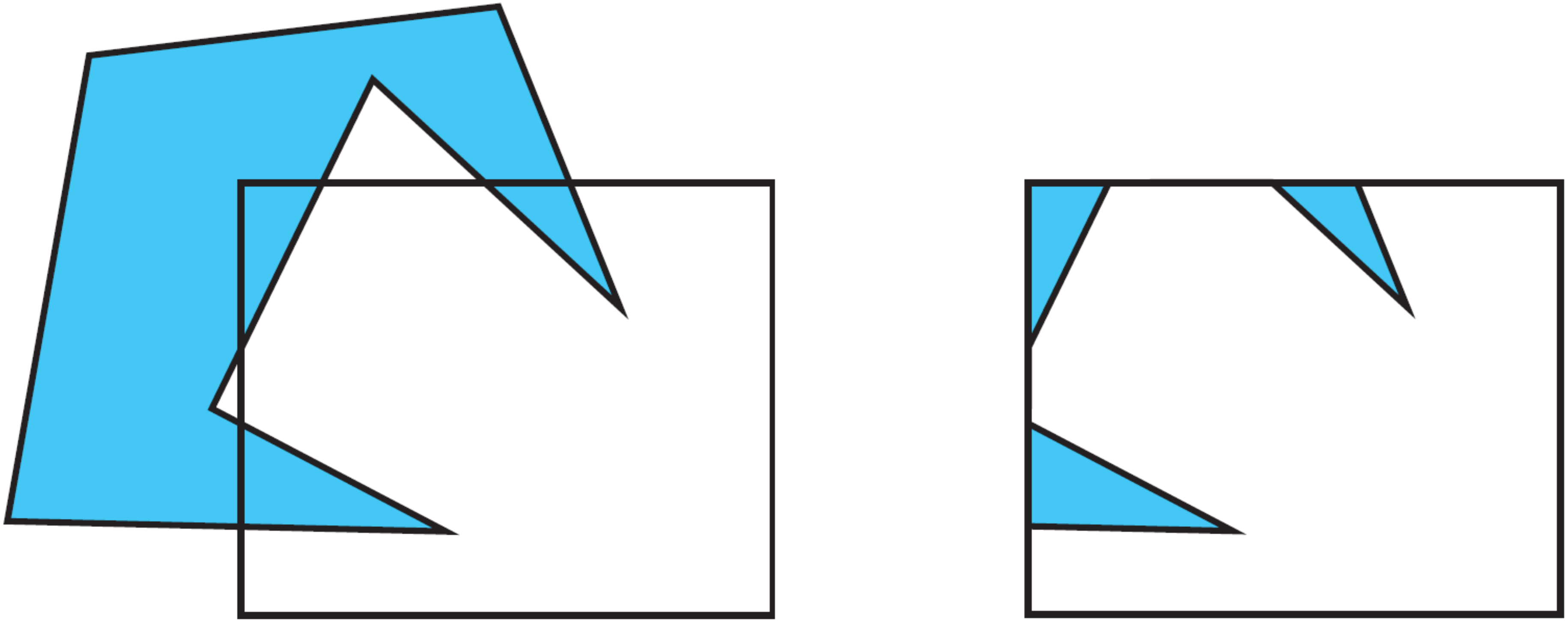
Can work with non-rectangular window

Other Line Clipping Algorithms

Liang-Barsky algorithm

Simplified Cyrus-Beck for rectangular window

Polygon Clipping



Polygon Clipping

Only vertices are given

No idea of fragments/pixels yet

Clipping → Rasterization

Sutherland–Hodgeman algorithm

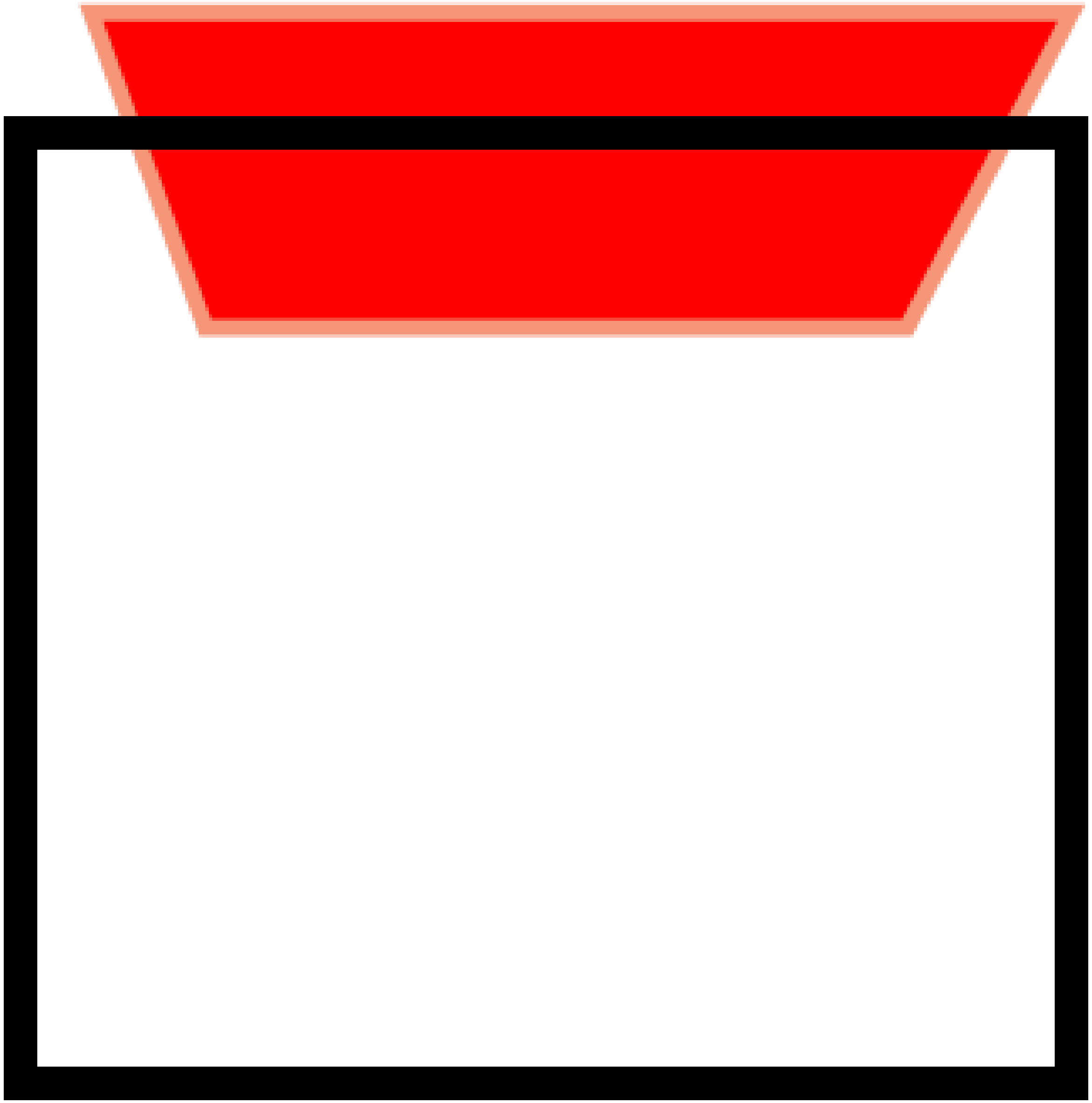
For all line segments of polygon from vertices

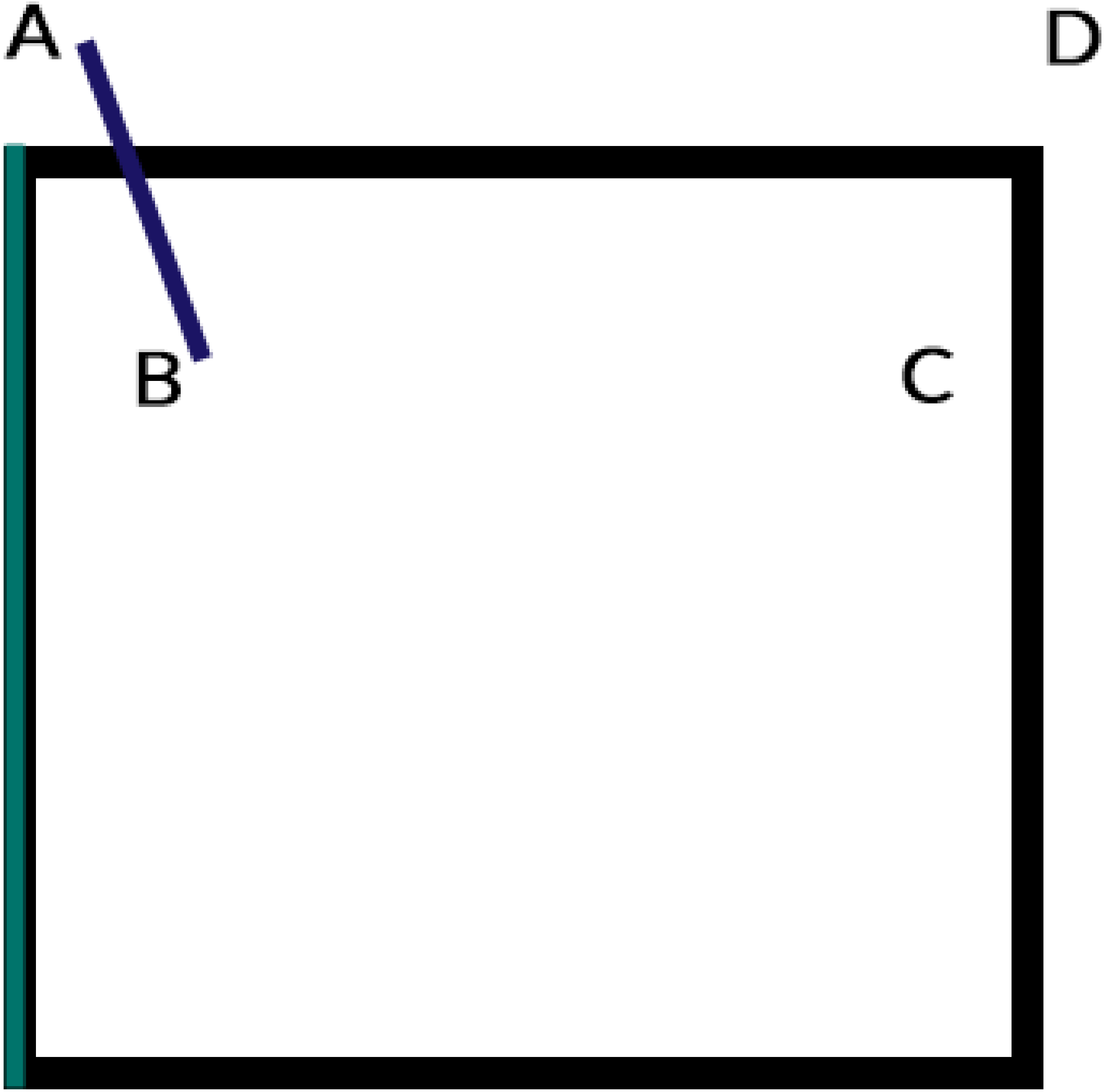
Clip all lines from left

Clip all lines from top

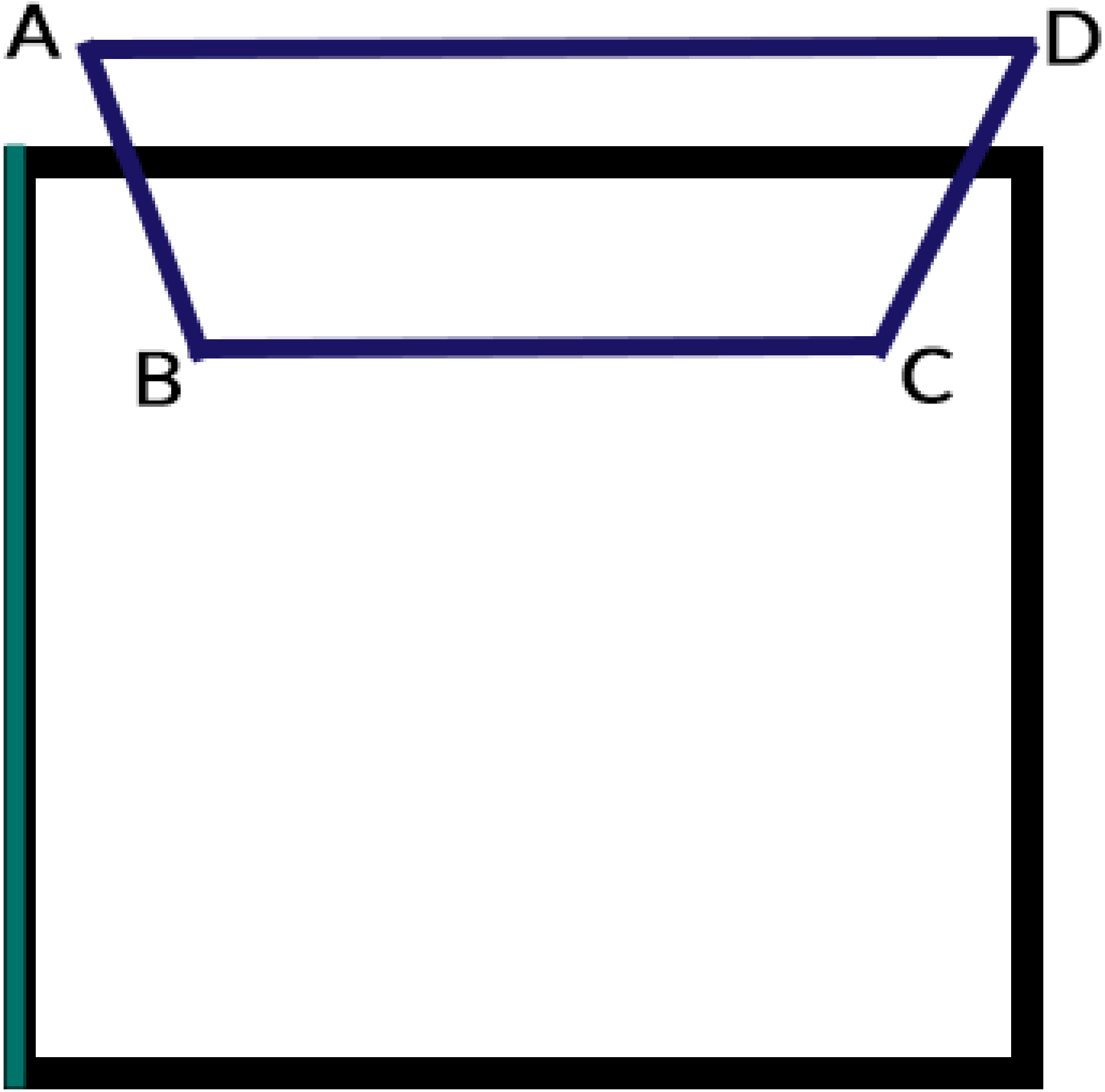
Clip all lines from right

Clip all lines from bottom

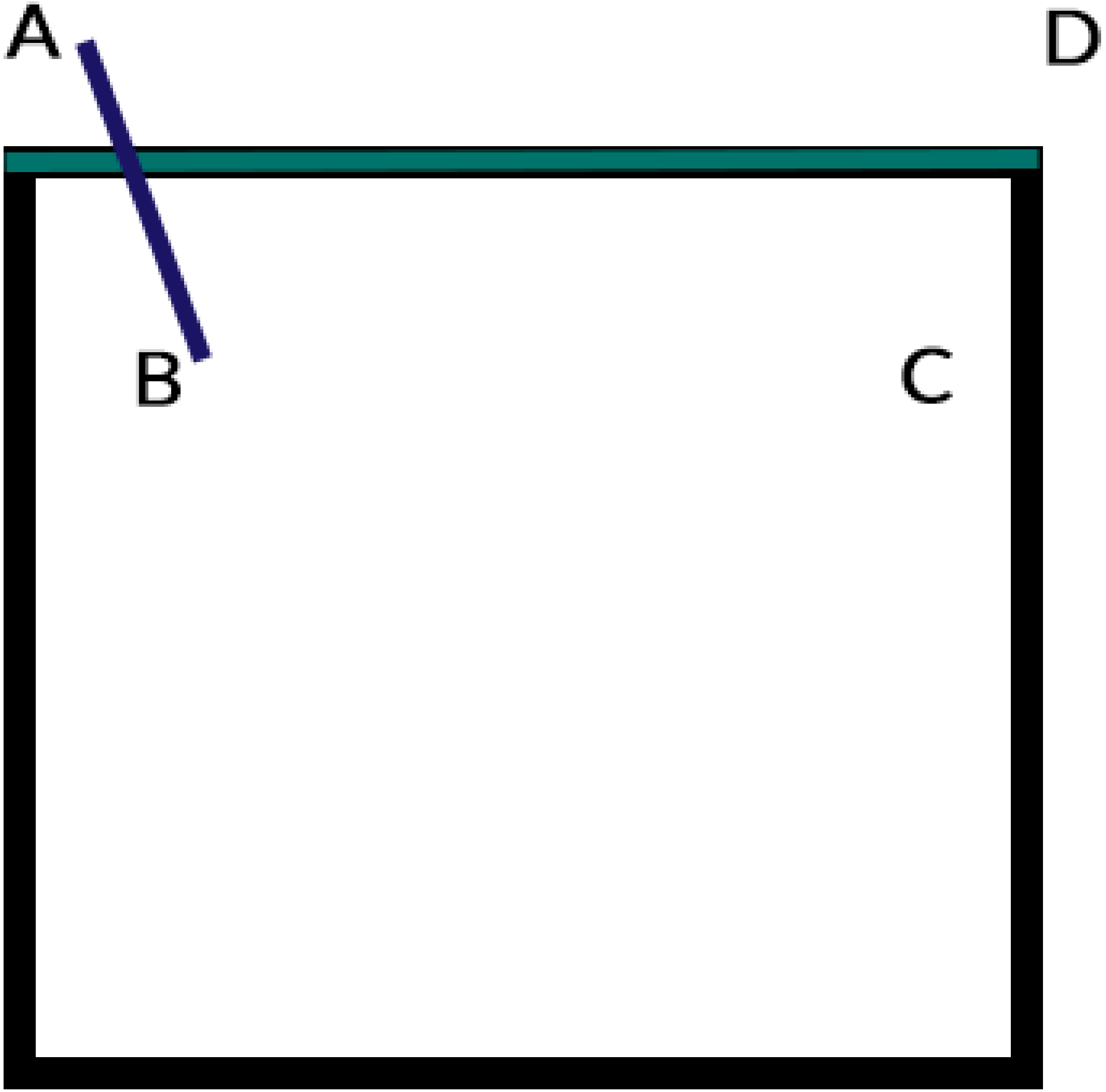




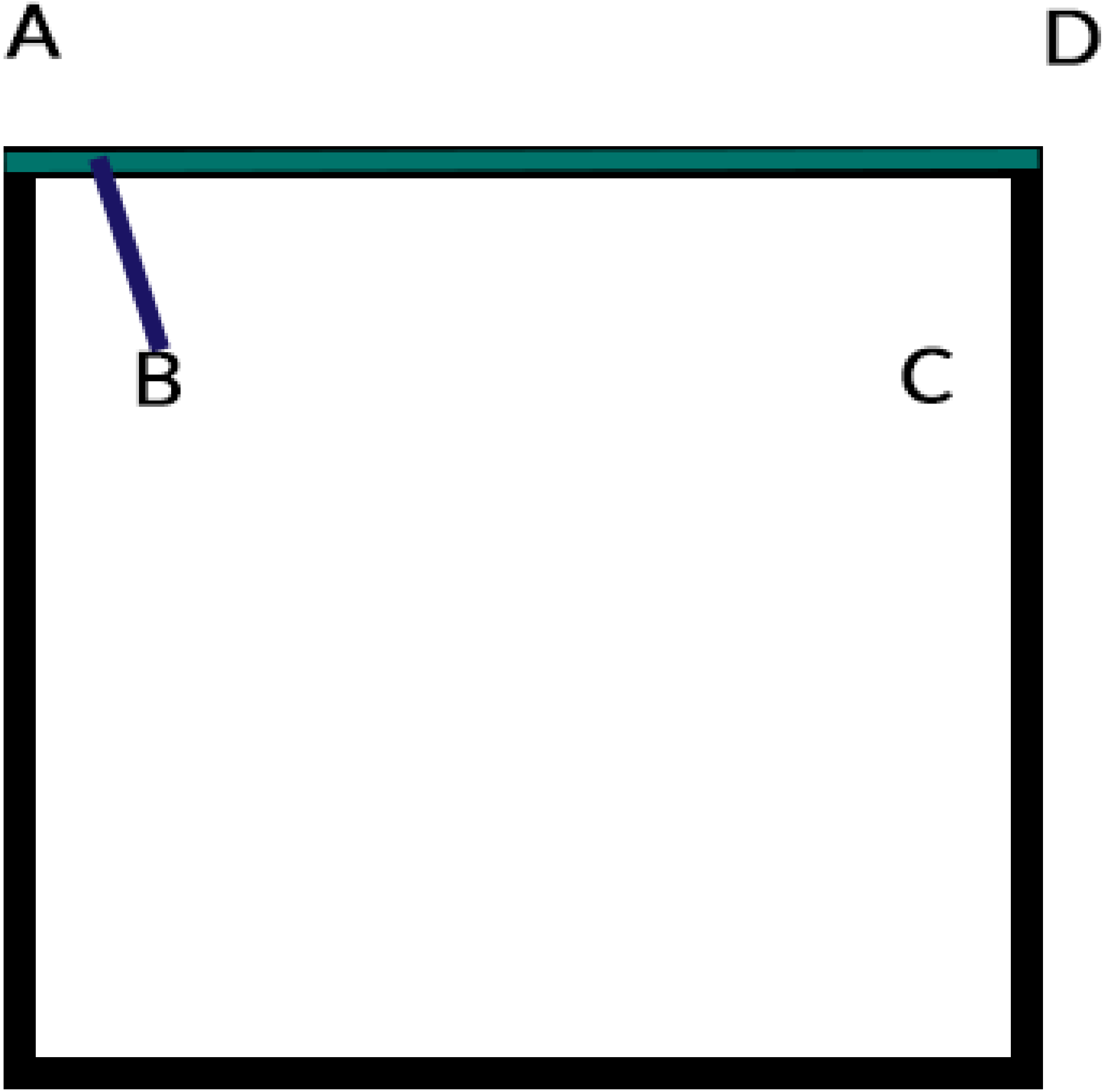
Clipping of line AB on left plane has no effect



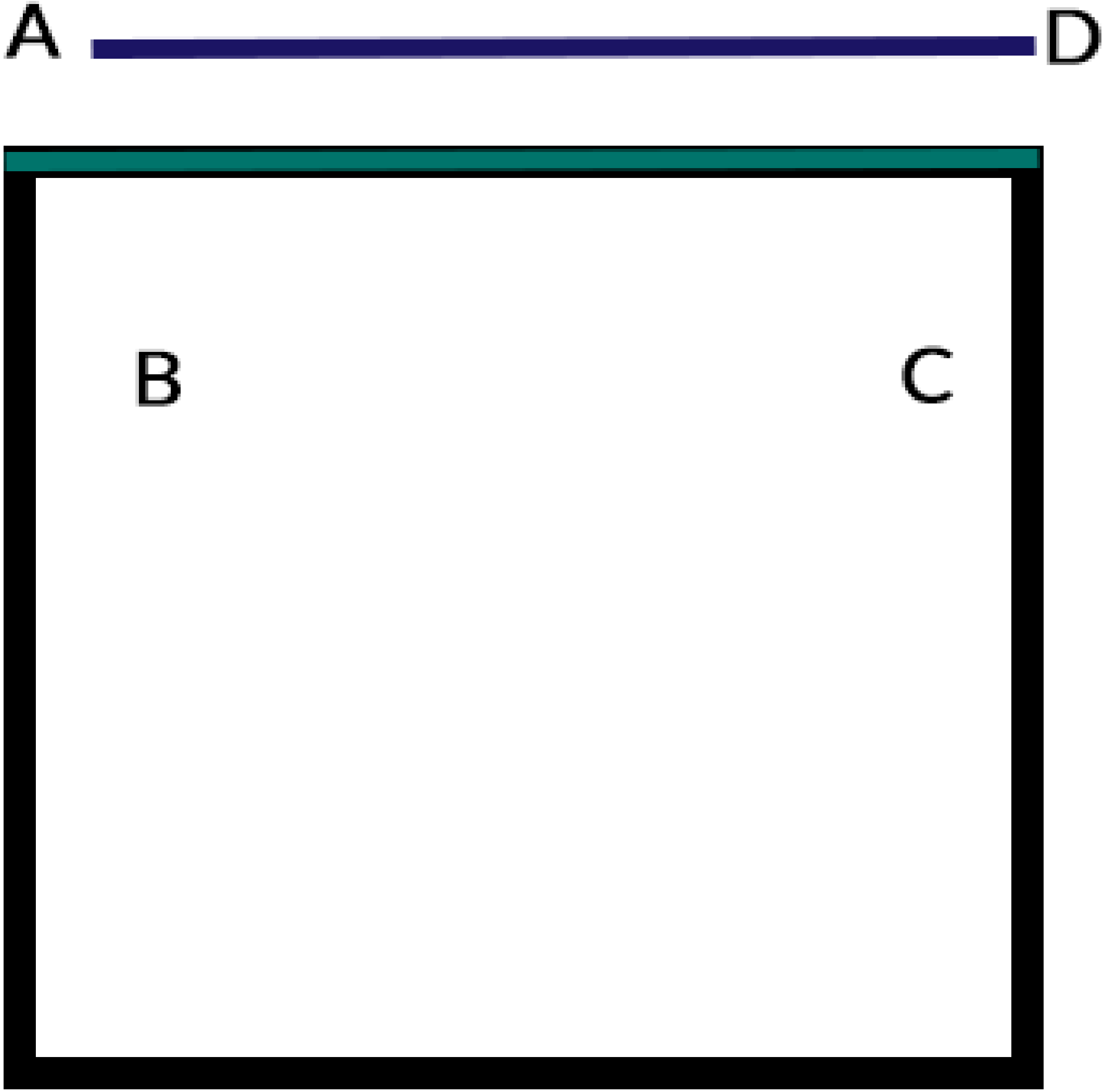
No clipping effect on left plane



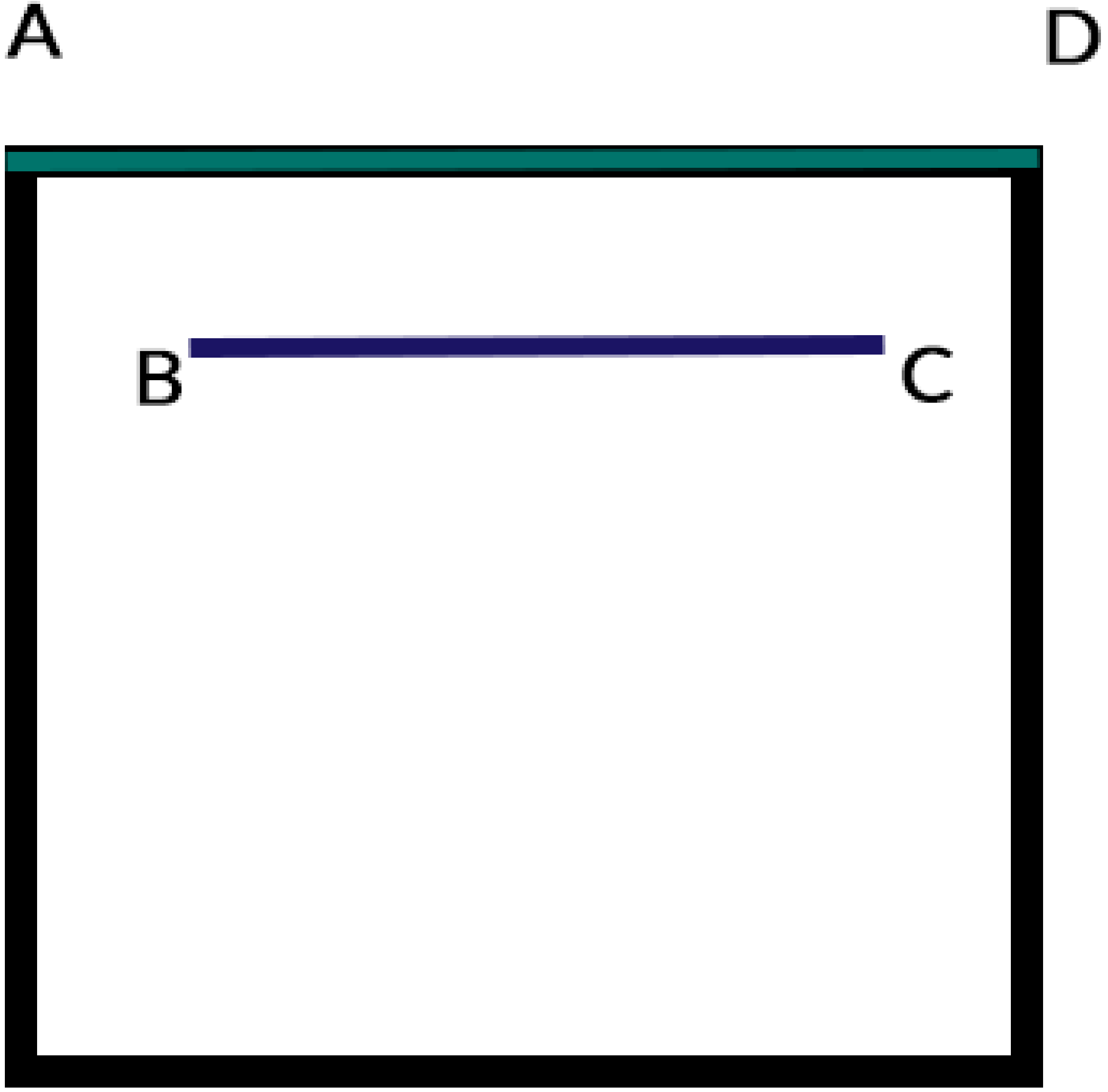
Clipping of line AB on top plane



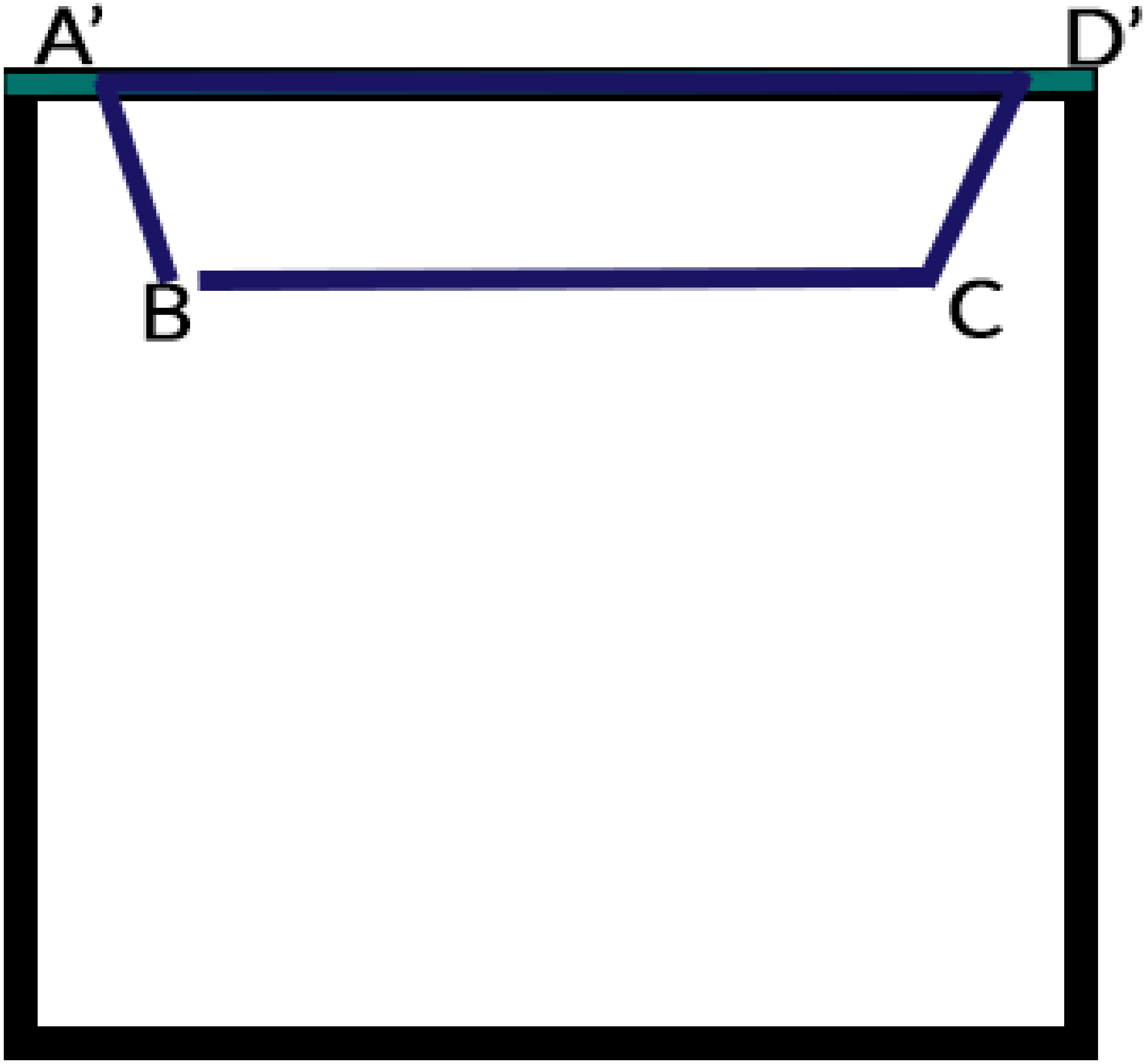
AB clipped using line clipping algorithms



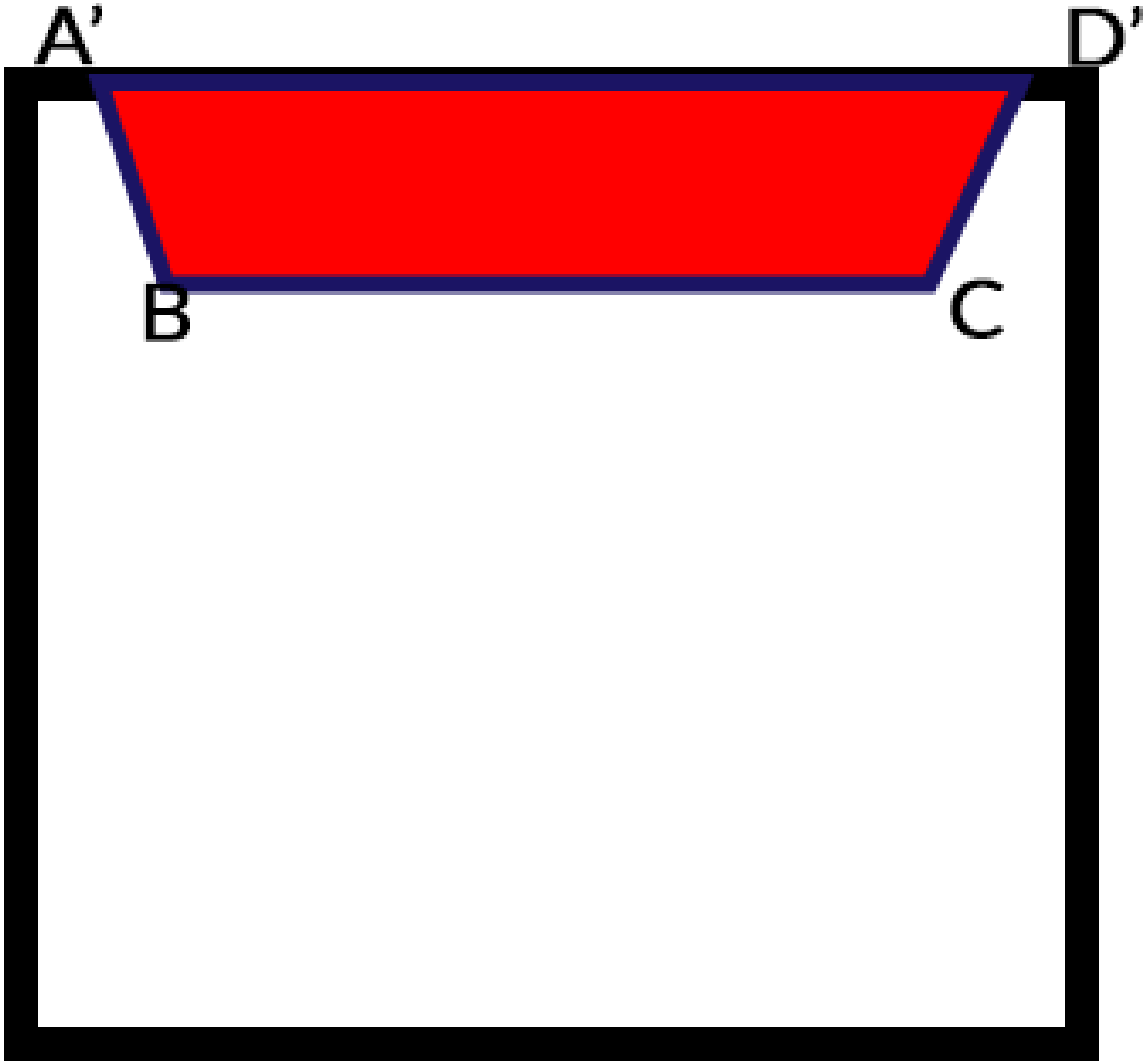
AD rejected



BC accepted



A’
D’
B
C


A’
D’
B
C

Sutherland–Hodgeman algorithm

Significantly faster than working with pixels

Works on primitive level (lines and points)

References

Books

- ANGEL, E. AND SHREINER, D. 2012. Interactive computer graphics : a top-down approach with shader-based OpenGL. Addison-Wesley. 6th ed. Boston, MA.
- CANTOR, D. AND JONES, B. 2012. WebGL Beginner's Guide. Packt Publishing. Birmingham, UK.
- MATSUDA, K. AND LEA, R. 2013. WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL.. Addison-Wesley. Upper Saddle River, NJ

Lecture Slides

- ALAMBRA, A. CMSC 161 1st Semester 2013-14 Lecture Slides

Images

- http://dev.opera.com/articles/view/raw-webgl-part1-getting-started/