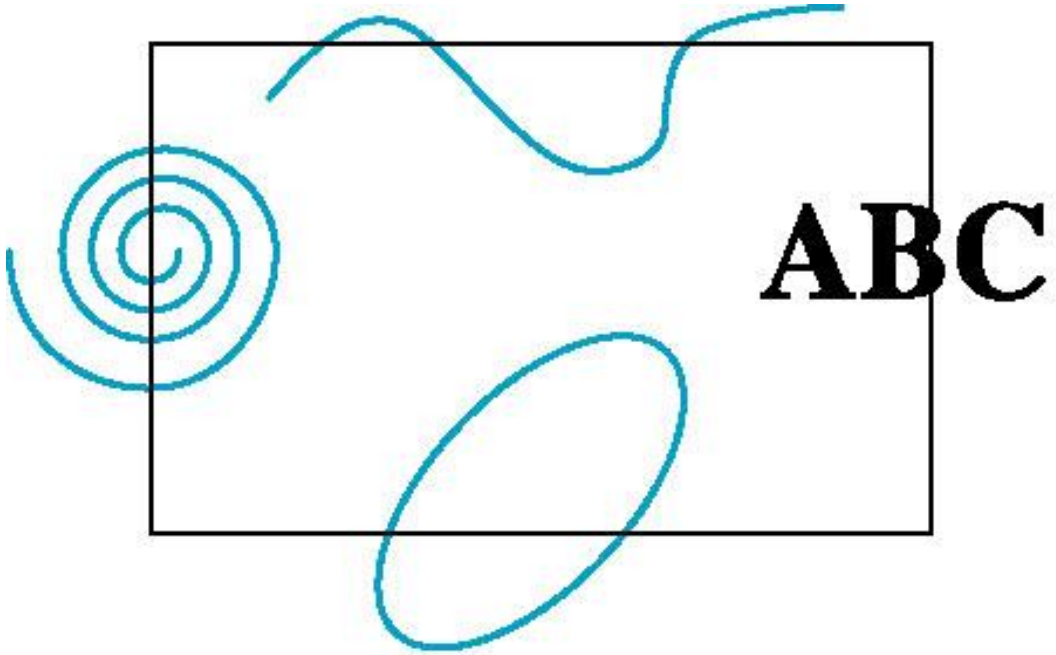
Clipping Primitives

- So far we have understood that all the object/image in the screen is consisting of pixels and specific a set of calculations is required for each pixel.
- In computer graphics, all primitives are clipped to the boundaries of this **clipping rectangle**; that is, primitives lying outside the clip rectangle are not drawn.
- The default clipping rectangle is the full canvas (the screen), and it is obvious that we cannot see any graphics primitives outside the screen.



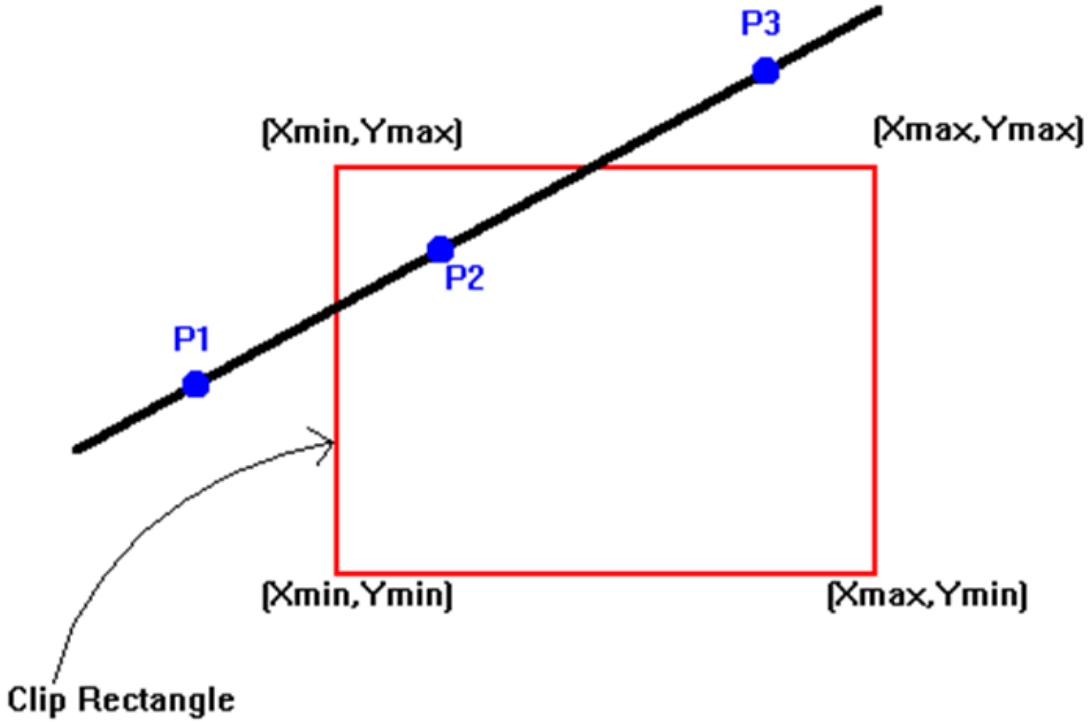
Clipping Individual Points

- Before we discuss clipping lines, let's look at the simpler problem of clipping individual points.
- If the x coordinate boundaries of the clipping rectangle are X_{min} and X_{max}, and the y coordinate boundaries are Y_{min} and Y_{max}, then the following inequalities must be satisfied for a point at (x,y) to be inside the clipping rectangle:

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

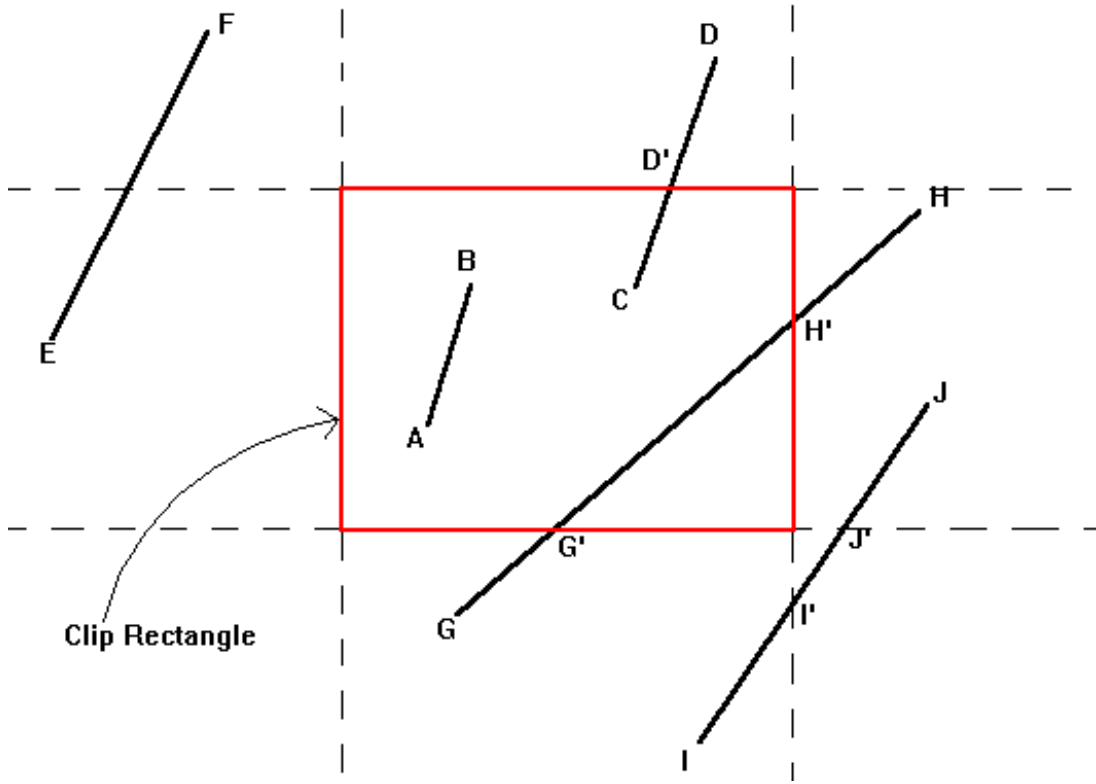
and $$Y_{min} \le y \le Y_{max}$$

- If any of the four inequalities does not hold, the point is outside the clipping rectangle.



Line Clipping

- To clip a line, we need to consider only its endpoints. If both endpoints of a line lie inside the clip rectangle (e.g., AB), the entire line lies inside the clip rectangle and can be **trivially accepted**.
- If one endpoint lies inside and one outside(e.g., CD), the line intersects the clip rectangle and we must compute the intersection point.
- If both endpoints are outside the clip rectangle, the line may or may not intersect with the clip rectangle (EF, GH, and IJ), and we need to perform further calculations to determine whether there are any intersections.



Cohen-Sutherland Line Clipping Algorithm

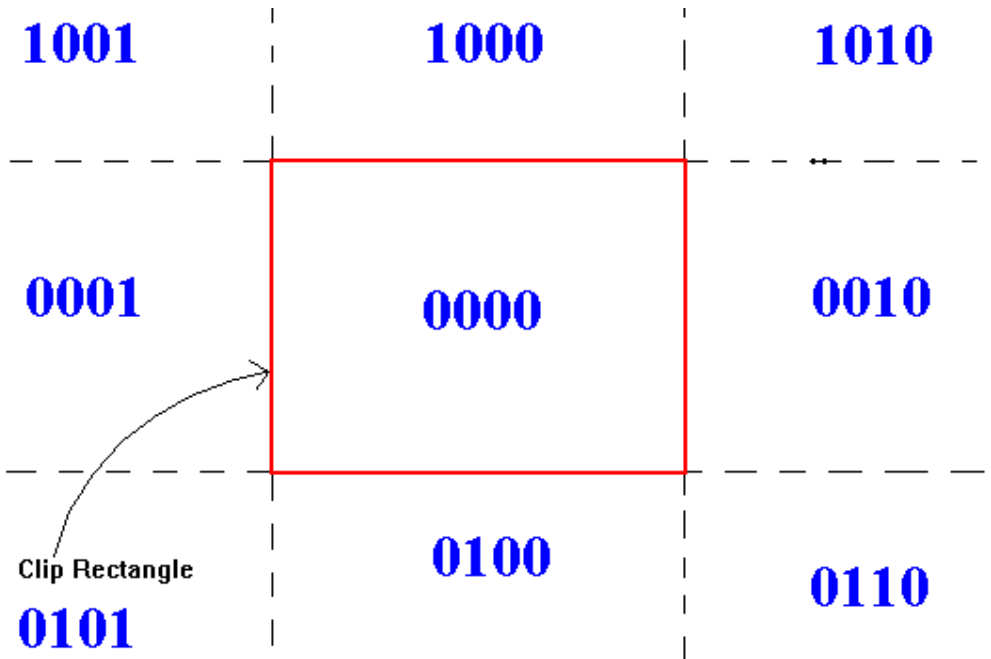
- Cohen-Sutherland Algorithm performs initial tests on a line to determine whether it is inside/outside clip region without intersection calculations.
- Steps for Cohen-Sutherland algorithm
  - End-points pairs are checked for trivial acceptance or trivial rejected using the region-outcode.
  - If not trivial-acceptance or trivial-rejected, divided into two segments at a clip edge.
  - Iteratively clip by testing trivial-acceptance or trivial-rejected, and divided into two segments until completely accepted or trivial-rejected.

Region Outcodes

- To perform trivial acceptance and rejection tests, we extend the edges of the clip rectangle to divide the plane of the clip rectangle into nine regions.
- Each region is assigned a 4-bit code determined by where the region lies with respect to the outside halfplanes of the clip-rectangle edges.
- Each bit in the outcode is set to either 1 (true) or 0 (false); the 4 bits in the code correspond to the following conditions

b_3	b_2	b_1	b_0

$b_3 = 1$ if $y > y_{max}$, 0 otherwise
$b_2 = 1$ if $y < y_{min}$, 0 otherwise
$b_1 = 1$ if $x > x_{max}$, 0 otherwise
$b_0 = 1$ if $x < x_{min}$, 0 otherwise



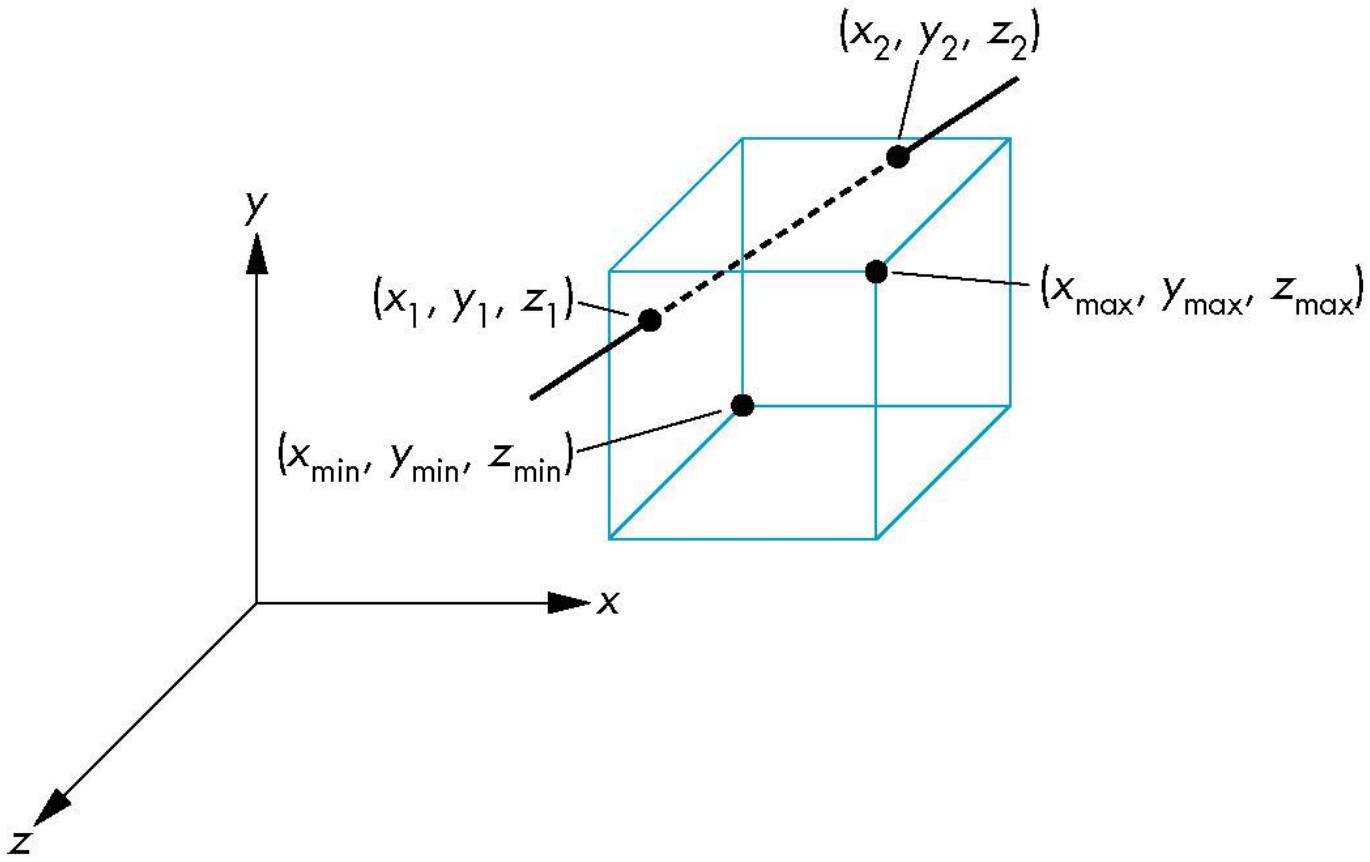
Algorithm for making Region Outcodes

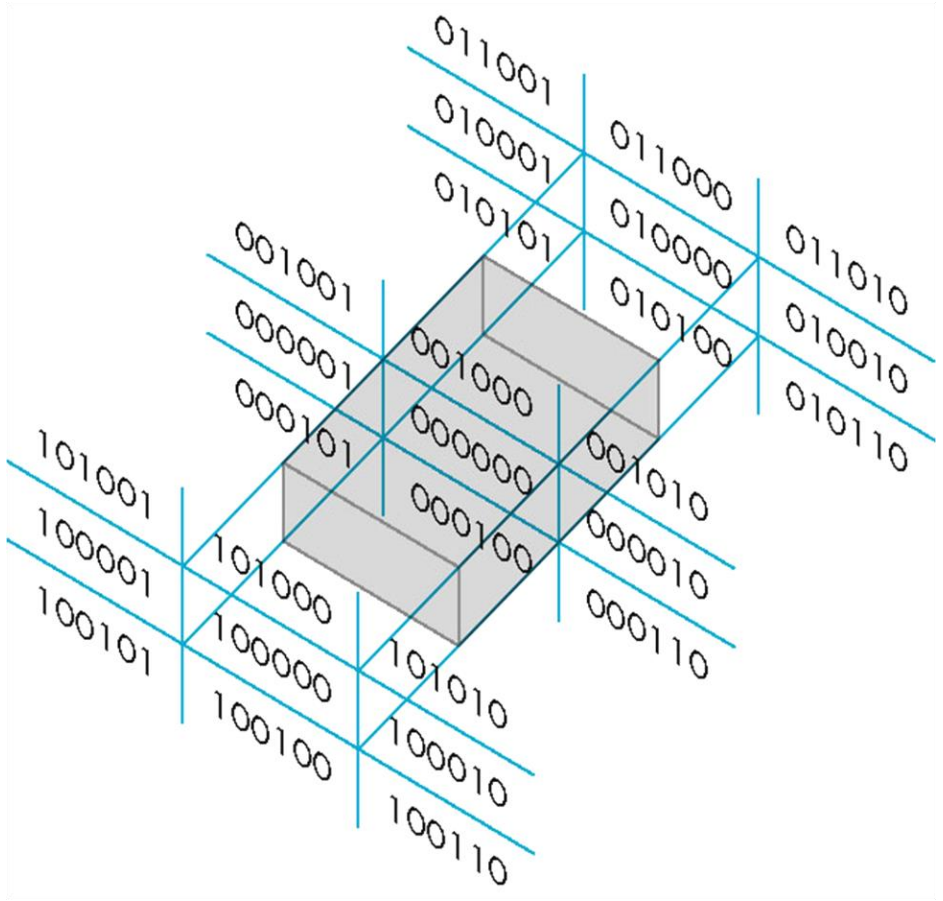
```
Xmax, Xmin, Ymax, Ymin = 319, -320, 239, -240
Left, Right, Bottom, Top = 1, 2, 4, 8
def OutCode (x, y):
        code = 0
        if y>Ymax:
                code  |=  Top
        elif y<Ymin:
                code  |=  Bottom

        if x>Xmax:
                code  |=  Right
        elif x<Xmin:
                code  |=  Left
        return code
code = OutCode(450, -400)
print(code)
```

Cohen Sutherland in 3D Clip-Region

- Use 6-bit outcodes, 2-bit for Z.
- Clip line segment against planes, left-right, bottom-top, far-near plane

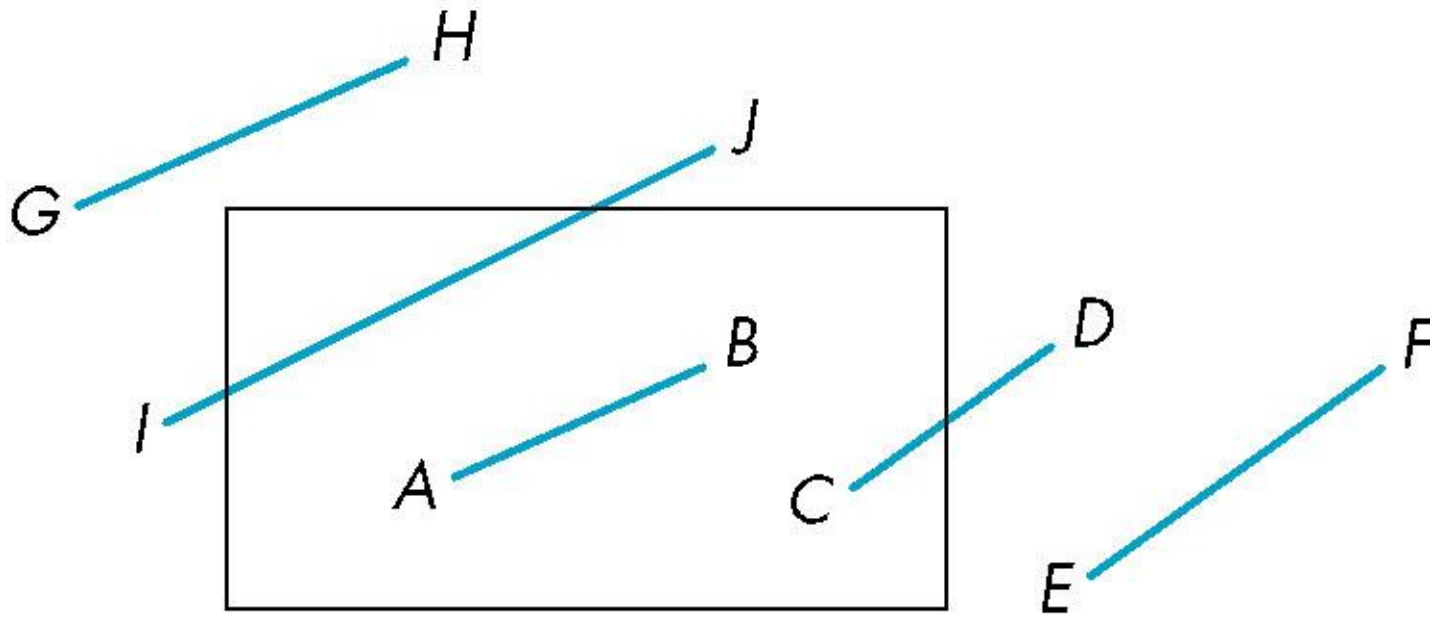




Decisions: Using Outcodes (Trivial acceptance)

Both endpoints are inside

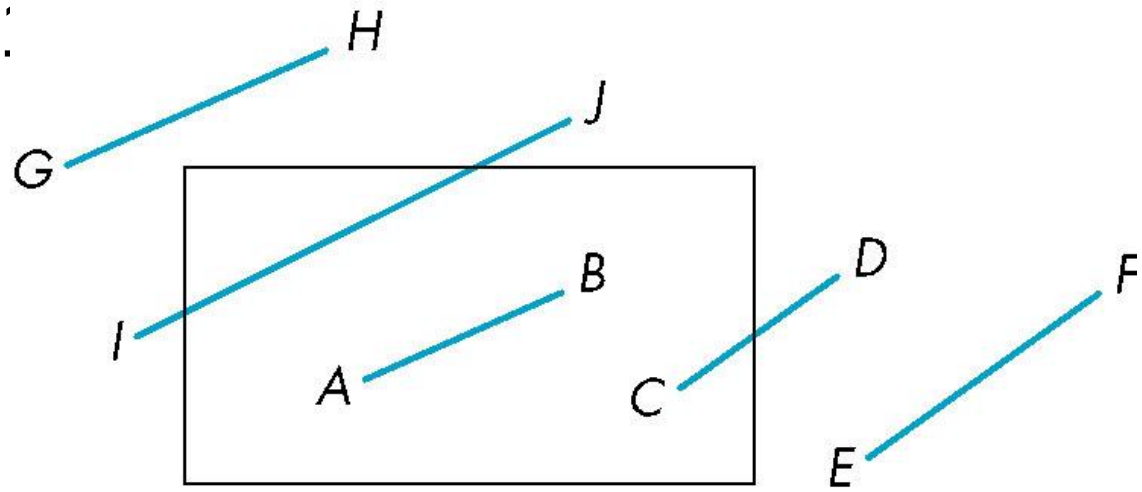
- If(both outcodes are zero) *Trivially accepted line*.
- Example: Line AB: outcode(A) = outcode(B) = 0000
  - *if(bitwise OR of the outcode of two endpoints == 0) →Accept the line*



Using Outcodes (Trivial Rejected)

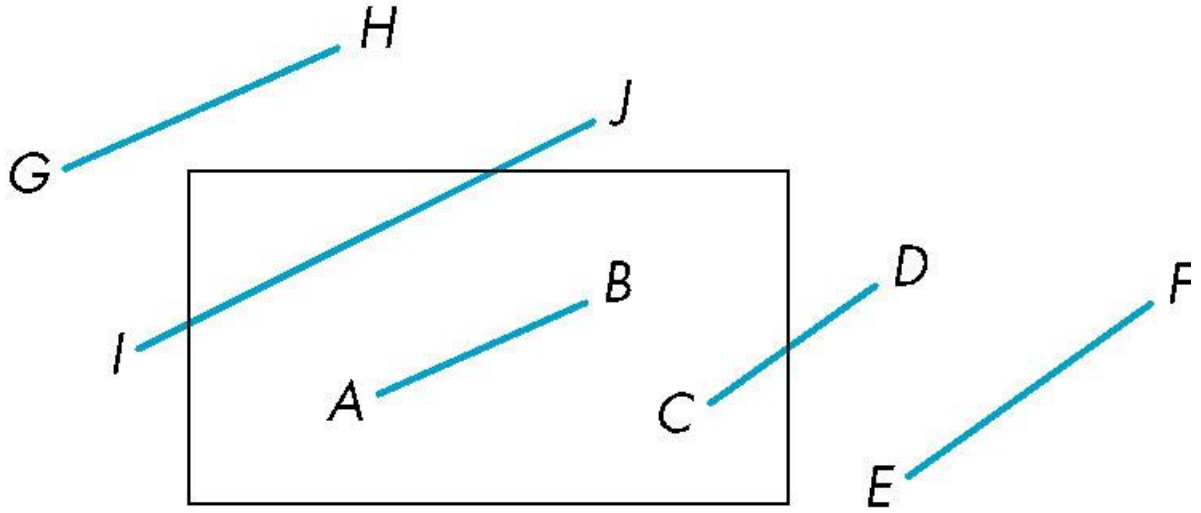
Both outcodes have a 1 bit in the same place

- Both endpoints of the line segment are in the same outside of clipping window
- *else-if(bitwise AND of the outcode of two endpoints ≠0) → rejected the line*.
- Example EF: outcode(E) 0110 logically (bitwise)ANDed with outcode(F) 0010 = 00
  - reject



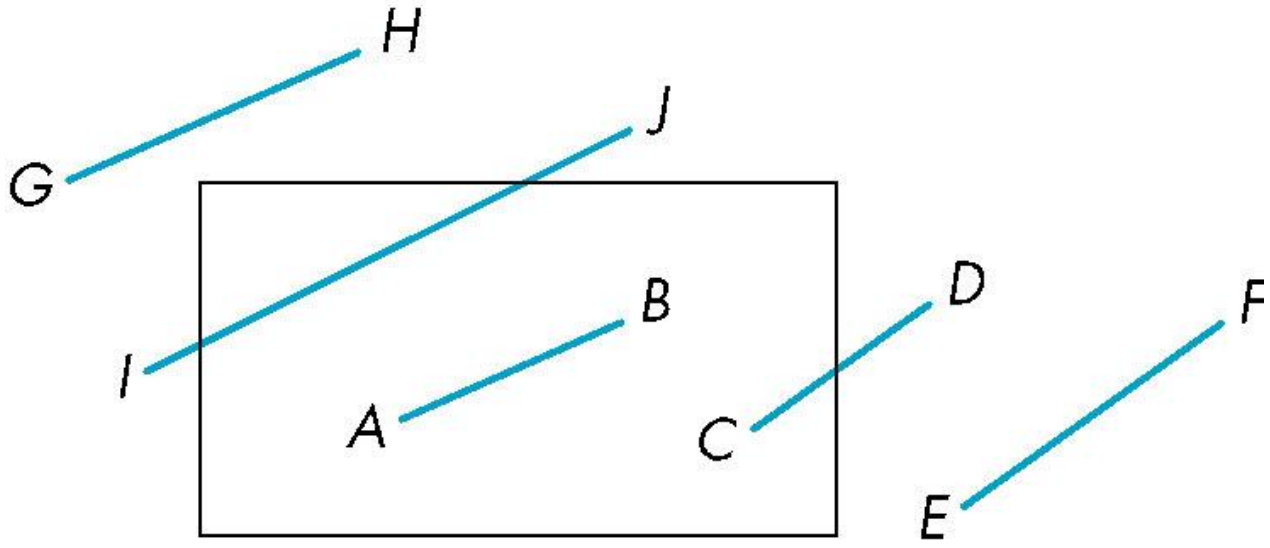
Using Outcodes (Partially Accepted/Rejected)

- CD: outcode (C) = 0000, outcode(D) = 0010
- IJ: outcode (J) = 0001, outcode(J) = 1000
  - Compute intersection
  - Location of 1 in outcode determines which edge to intersect with (D is in the right-side, J in the top)
  - Note if there were a segment from A to a point in a region with 2 ones in outcode (e.g., E), we might have to do two intersections



Using Outcodes (Partially Accepted/Rejected)

- GH and IJ: same outcodes, neither zero but logical AND yields zero
- Shorten line segment by intersecting with one of sides of window, i.e., reduce 1 from the outcode
- Recompute outcode of intersection (new endpoint of shortened line segment)
- Re-execute algorithm until accepted/rejected (IJ partially accepted and GH rejected)



The Complete Program

```
def CohenSutherland(x0, y0, x1, y1):
    outcode0 = OutCode(x0, y0)
    outcode1 = OutCode(x1, y1)
    while 1:
        if (outcode0 | outcode1) == 0: # Trivial accept and exit
            print("X0=",int(x0),"Y0=", int(y0),"X1=", int(x1), "Y0=",int(y1))
            break
        elif (outcode0 & outcode1) != 0: # Trivial reject and exit.
            print ("Rejected\n")
            break
        else:                                   # Failed both tests, so calculate the line segment to clip;
            if outcode0 != 0:
                outcode = outcode0
            else:
                outcode = outcode1
            if Top & outcode:                   # Now find intersection point;
                y = Ymax
                x = x0 + (x1 - x0) * (Ymax - y0) / (y1 - y0)
            elif Bottom & outcode:
                y = Ymin
                x = x0 + (x1 - x0) * (Ymin - y0) / (y1 - y0)
            elif Right & outcode:
                x = Xmax
                y = y0 + (y1 - y0) * (Xmax - x0) / (x1 - x0)
            else:                               #Left & outcode
                x = Xmin
                y = y0 + (y1 - y0) * (Xmin - x0) / (x1 - x0)
            if outcode == outcode0: # One outside point moved to boundary, get ready for the next
                x0 = x
                y0 = y
                outcode0 = OutCode(x0, y0)
            else:
                x1 = x
                y1 = y
                outcode1 = OutCode(x1, y1)
CohenSutherland(-500, -123, 400, 130)
```

Efficiency

- In many applications, the clipping window is small relative to the size of the entire data base
  - Most line segments are outside one or more side of the window and can be eliminated based on their outcodes
- Inefficiency when code has to be re-executed for line segments that must be shortened in more than one step (worst case 4 times re-execution)

Possible Quiz

Xmin	Ymin	Zmin	Xmax	Ymax	Zmax
-120	-100	-100	120	100	100

	X_0	Y_0	Z_0	X_1	Y_1	Z_1	Code0	Code1	Comments
1	-100	100	93	119	-96	81	0 0 0 0 0 0	0 0 0 0 0 0	Accepted
2	132	-101	-120	-121	150	113	0 1 0 1 1 0	1 0 1 0 0 1	Partial
3	231	111	-87	131	-10	-120	0 0 1 0 1 0	0 1 0 0 1 0	Rejected
4	120	100	79	-100	-100	-80	0 0 0 0 0 0	0 0 0 0 0 0	Accepted