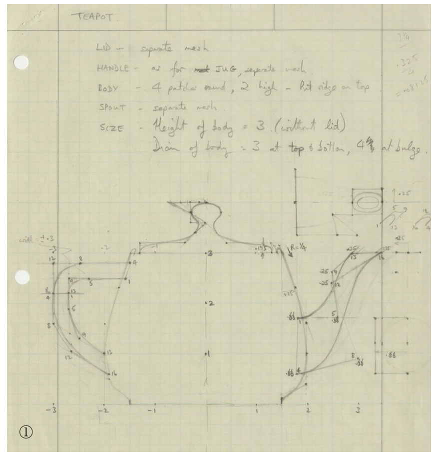
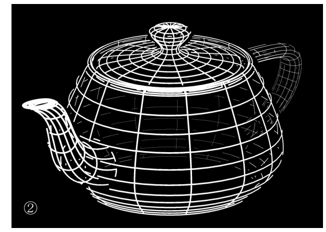
Computer Graphics





CSE 4303 / CSE 5365
Clipping, 2019 Fall

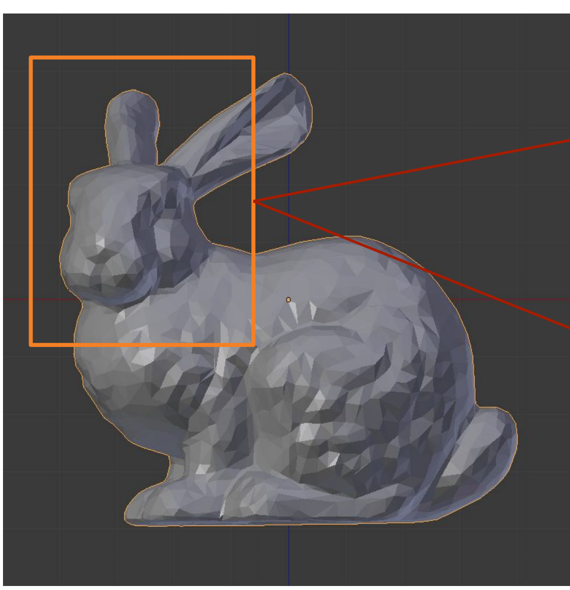
① http://www.computerhistory.org/revolution/computer-graphics-music-and-art/15/206/556
② http://www.cs.technion.ac.il/~gershon/site/img/gallery/gallery-pic-cat3-depth-cueing-2-big.jpg
③ http://www.omnigraphica.com/gallery/maingallery/original/Utah_teapot_1.png
④ http://unfold.be/assets/images/000/113/719/large-utanalog3.jpg

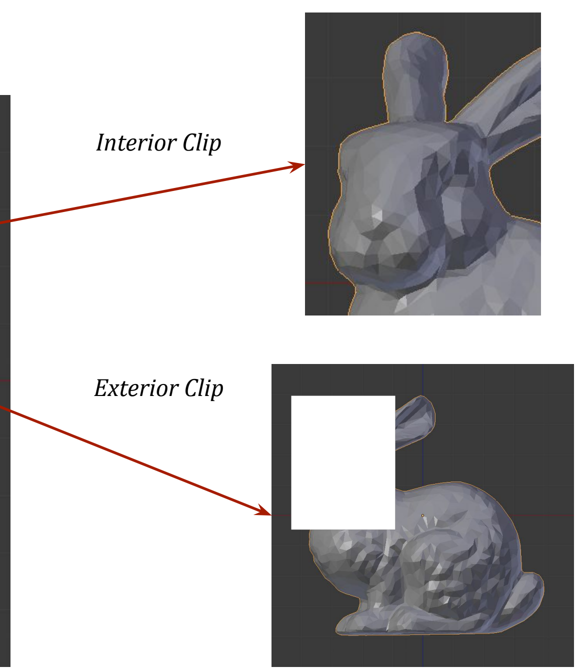
Clipping

Clipping

- *Clipping* is the identification of objects or parts of objects as either *inside* or *outside* a specified region.
- *Interior* clipping is the saving of what's *inside* the region.
  - For example, *copy* a piece of a picture.
- *Exterior* clipping is the saving of what's *outside* the region.
  - For example, *clear* a piece of a picture

Clipping





Clipping

- In CG, clipping is primarily used to decide which objects or *parts* of objects should be visible when a scene is rendered.

- Why clip?
  - Don't waste time on objects that can't be seen.
    - Or even an unseeable *part* of an object.
  - Avoid degenerate cases that might cause divide-by-0 or overflow conditions.
    - As when a point is *behind* the camera.

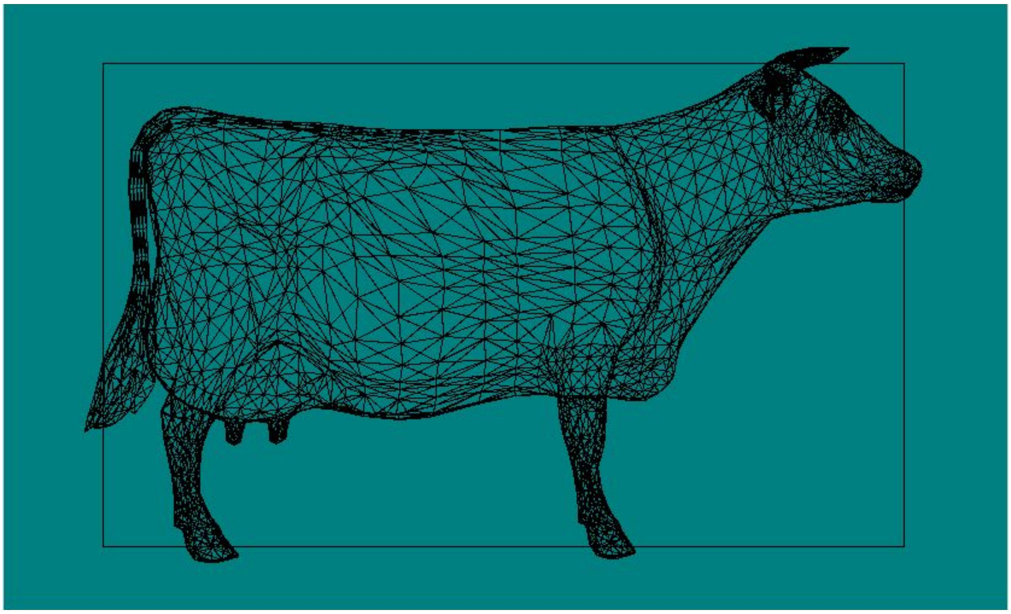
Clipping

- Clipping may be done at various points in the rendering pipeline.
  - Each point has its own way to specify the clipping region.
  - In 3D, it's a volume. In 2D, it's a region, usually rectangular.

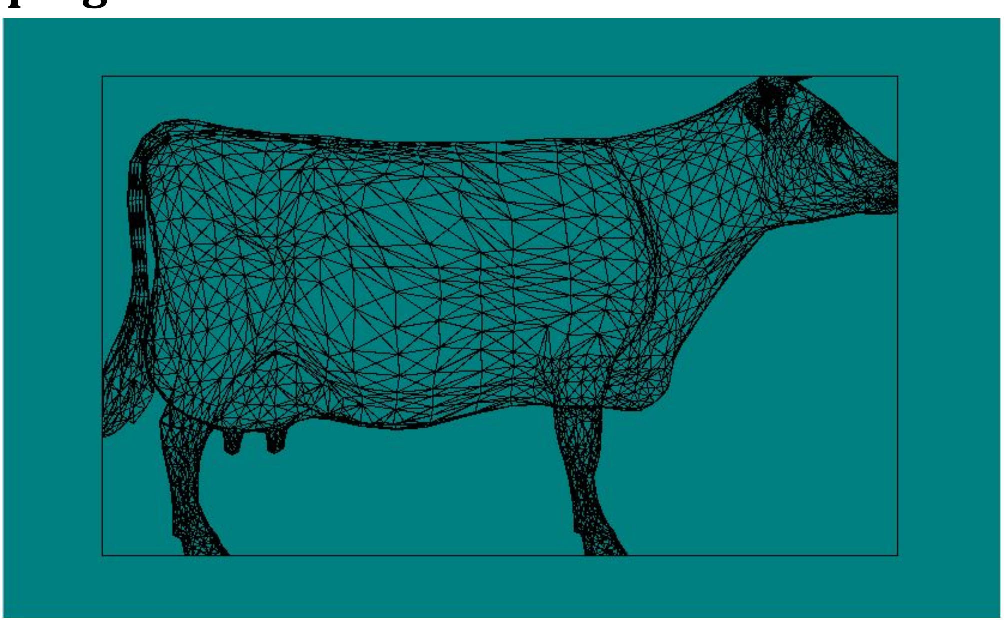
- Different kinds of clipping include
  - *Point*: Keep point only if inside.
  - *Line*: Keep portion of line that's inside, if any.
  - *Polygon*: Make a new polygon that's the portion inside, if any.

- Since we are drawing lines at present, we will start with 2D clipping of lines against the view window.

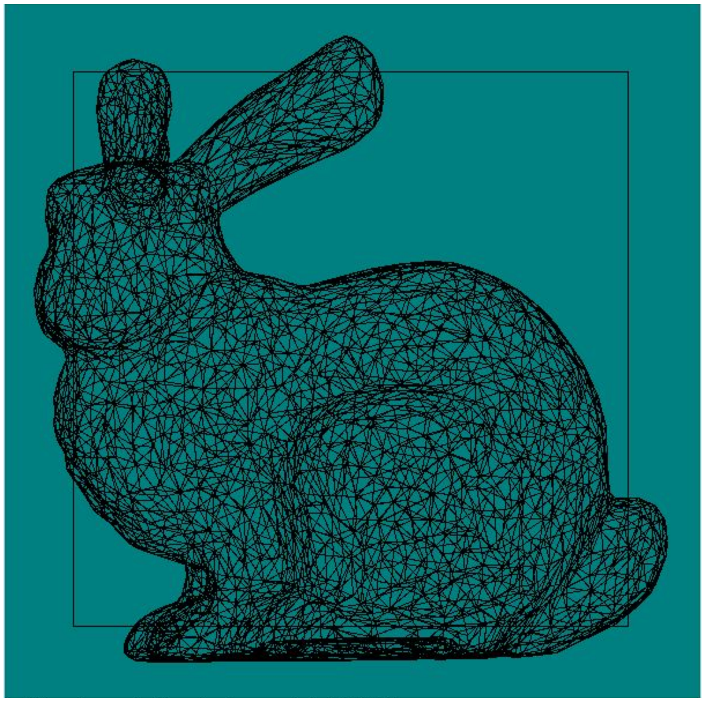
Clipping



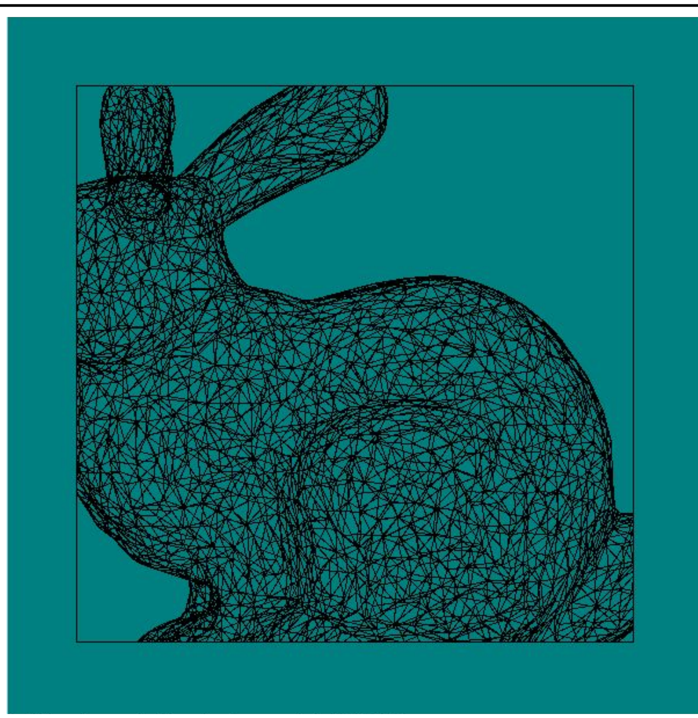
Clipping



Clipping

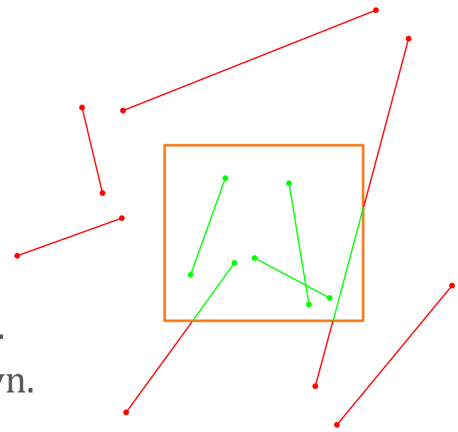


Clipping



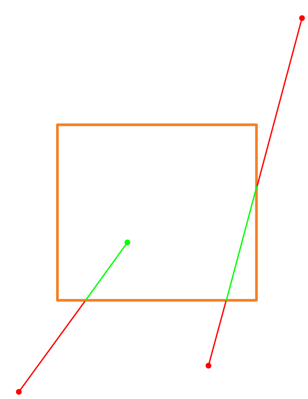
Clipping

- Consider a viewport that we want to clip against.
- Some lines are clearly *inside* the region and should be drawn.
- Some lines are clearly *outside* the region and should *not* be drawn.
- Others are both inside *and* outside.
- *Parts* of those lines should be drawn.



Clipping

- Notice there are two kinds of *partial* lines.
  - One of the points is *inside* the clipping region.
  - Both of the points are *outside* the clipping region.
- We cannot eliminate a line just because both of its points are *outside*.



Line Clipping

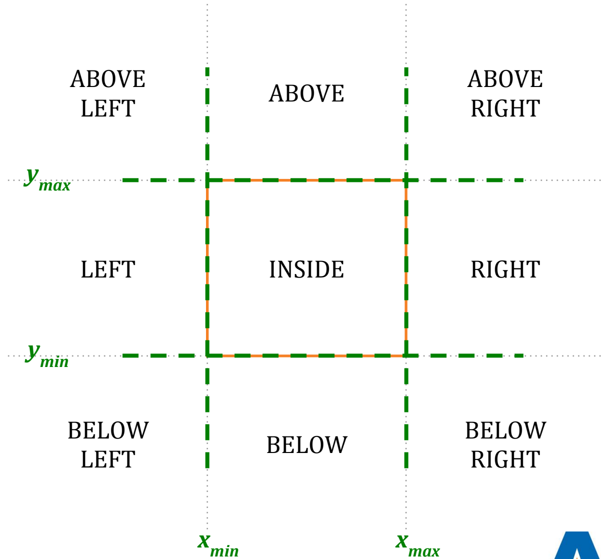
- There are many, many methods for line clipping.
  - They all have various claims to fame, application area, capability, speed, simplicity, etc.
- (One of) the earliest is the Cohen-Sutherland method.
  - Invented by Danny Cohen and Ivan Sutherland in 1967 while working on a flight simulator.
- It's a simple algorithm.
  - Quickly accepts completely inside lines. Quickly rejects certain categories of completely outside lines.
  - Uses iteration to make decisions on the rest.

[*Quick Drawing Review*]

- Objects are defined as a set of vertices and faces.
  - The `v` x y z lines specify the vertex's position in world space.
  - The `f` v_1 v_2 v_3 lines specify which vertices make up each face.
- The positions of the vertices are transformed from x, y, z world coordinates into x, y pixel coordinates.
- Three lines are drawn for each face.
  - ① v_1 to v_2 ② v_2 to v_3 ③ v_3 to v_1
- Because a vertex may end up outside the viewport region, each of these lines may need to be clipped.

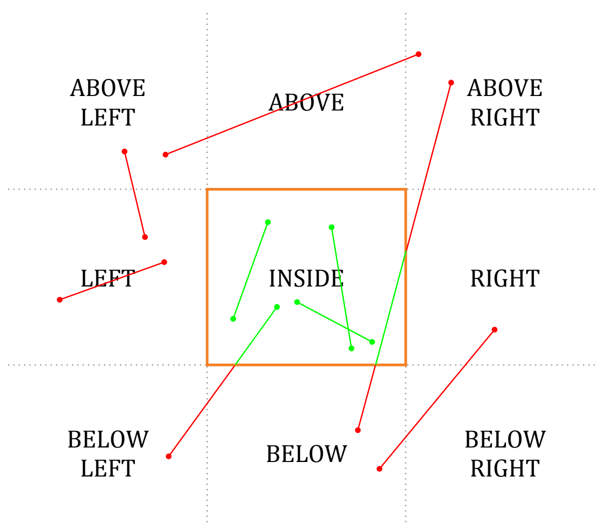
Cohen-Sutherland Line Clipping

- Divides the viewport space into nine areas.
- The central area is the *inside* space that the user sees.
- All other areas are *outside* and are not seen.
- INSIDE is bounded by the lines x_{min}, x_{max}, y_{min}, y_{max}.



Cohen-Sutherland Line Clipping

- Step one in clipping a line is to determine in which of the nine regions its end points fall.
- This is easy! :)



Cohen-Sutherland Line Clipping

- Starting with the point's x and y coordinates …
- Compare x against x_{min} and x_{max} to determine if the point is LEFT or RIGHT.
- Compare y against y_{min} and y_{max} to determine if the point is BELOW or ABOVE.
- Done! :)

```
INSIDE = 0
LEFT   = 1
RIGHT  = 2
BELOW  = 4
ABOVE  = 8
```

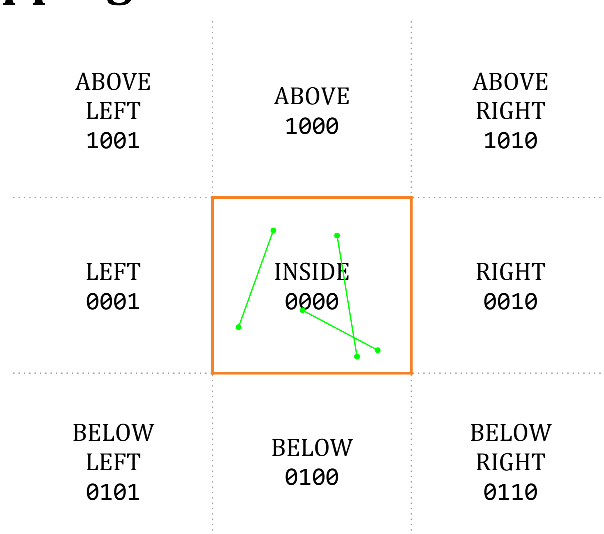
These are mutually exclusive powers of 2, so each is a unique bit.

```
def outcode( x, y, xMin, yMin, xMax, yMax ) :
 code = INSIDE

 if ( x < xMin ) :
   code = code | LEFT
 elif ( x > xMax ) :
   code = code | RIGHT

 if ( y < yMin ) :
   code = code | BELOW
 elif ( y > yMax ) :
   code = code | ABOVE

 return code
```

Bit-wise OR *operations, so no bit interferes with another..*

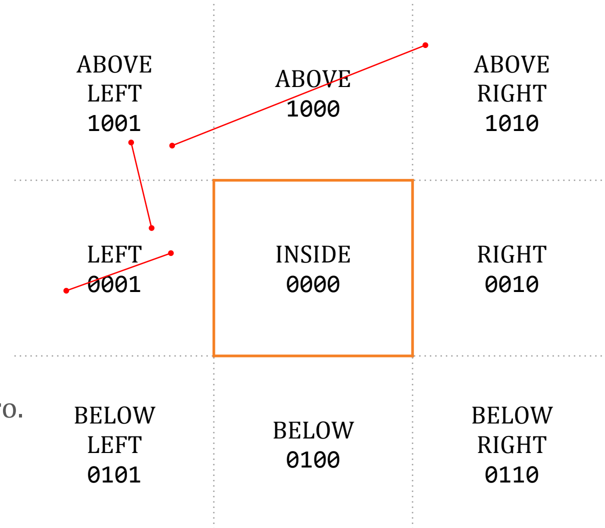
Cohen-Sutherland Line Clipping

- The result will be a 4-bit code corresponding to the area in which the point is.
- Notice that INSIDE's code ends up being `0000`.
- This makes it trivial to accept a line that is completely INSIDE.
- Both points will have code `0000`.
- Easy to detect!



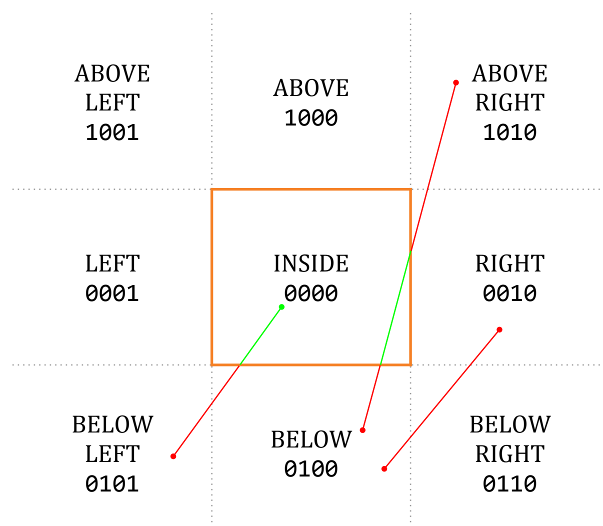
Cohen-Sutherland Line Clipping

- What about trivial rejects?
- Lines whose points are both in the same region that is *not* INSIDE can be rejected.
  - Line is entirely in unseen region.
- Lines whose points are on the same side of INSIDE can be rejected.
  - Line cannot intersect INSIDE region so nothing to draw.
- How to compute these relationships?
  - Bitwise AND of codes will be non-zero.
- Easy to detect!



Cohen-Sutherland Line Clipping

- What about the *mixed* or *both outside* cases?
- We have to determine which *portion* of the line *if any* is to be drawn.
- The algorithm is relatively simple, sliding one point or the other along the line to its bounding line.



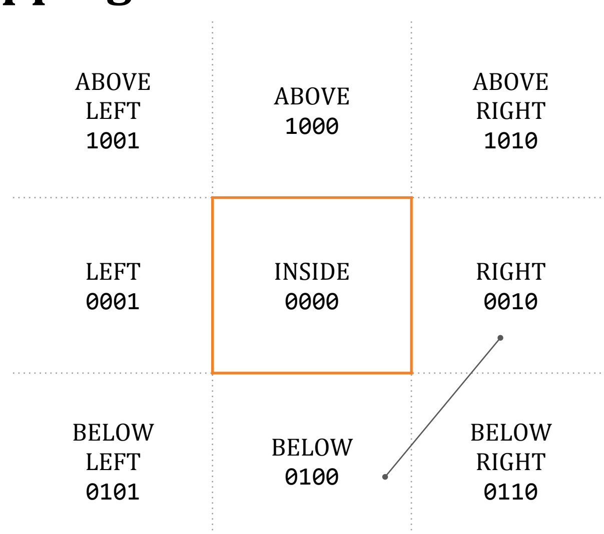
Cohen-Sutherland Line Clipping

- Pick a point that is *not* INSIDE.
  - There has to be one, otherwise the line would be a trivial accept.
- Slide that point to the spot on the line that removes (one of) its out-of-bounds problems.
  - If ABOVE, slide to y_{max} along the line.
  - If BELOW, slide to y_{min} along the line.
  - If RIGHT, slide to x_{max} along the line.
  - If LEFT, slide to x_{min} along the line.
- Even after sliding one point, the line might still be non-trivial to accept or reject, so iterate.
  - Replace the point with the new x, y and recompute its code first.

```
p1Out = outcode( p1x, p1y, xMin, yMin, xMax, yMax )
p2Out = outcode( p2x, p2y, xMin, yMin, xMax, yMax )

anOutCode = p2Out if p1Out == INSIDE else p1Out

if ( anOutCode & ABOVE ) :
 # Move point along the line down to Y max.
 x = p1x + ( p2x - p1x )*( yMax - p1y )/( p2y - p1y )
 y = yMax

elif ( anOutCode & BELOW ) :
 # Move point along the line up to Y min.
 x = p1x + ( p2x - p1x )*( yMin - p1y )/( p2y - p1y )
 y = yMin

elif ( anOutCode & RIGHT ) :
 # Move it along the line over to X max.
 x = xMax
 y = p1y + ( p2y - p1y )*( xMax - p1x )/( p2x - p1x )

elif ( anOutCode & LEFT ) :
 # Move it along the line over to X min.
 x = xMin
 y = p1y + ( p2y - p1y )*( xMin - p1x )/( p2x - p1x )
```

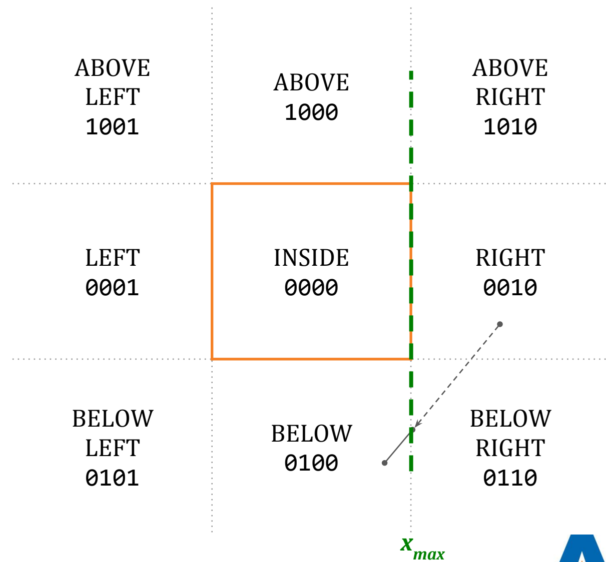
Cohen-Sutherland Line Clipping

- For example, both points of this line are outside the INSIDE region.



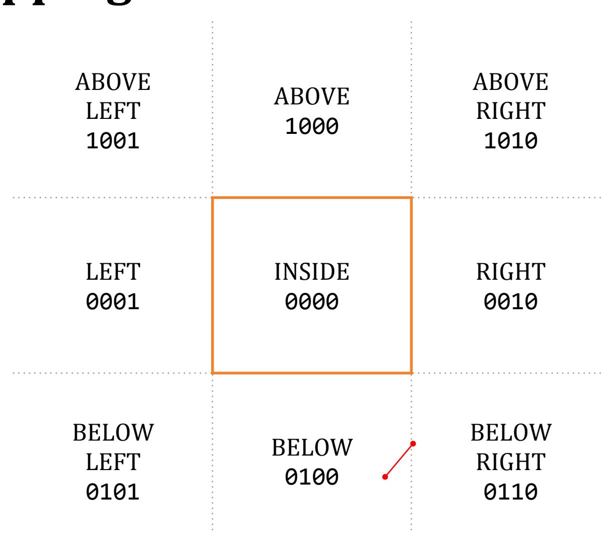
Cohen-Sutherland Line Clipping

- For example, both points of this line are outside the INSIDE region.
- If we manipulate the point in RIGHT, it gets slid to the x_{max} line.



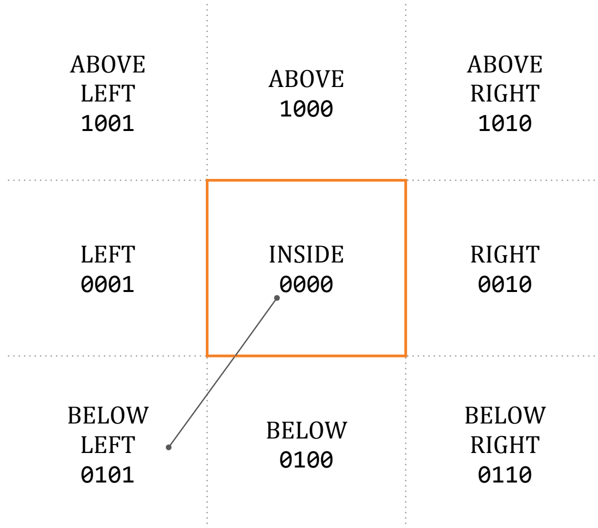
Cohen-Sutherland Line Clipping

- For example, both points of this line are outside the INSIDE region.
- If we manipulate the point in RIGHT, it gets slid to the x_{max} line.
- When we recompute its code, both points will be BELOW, so trivial reject.



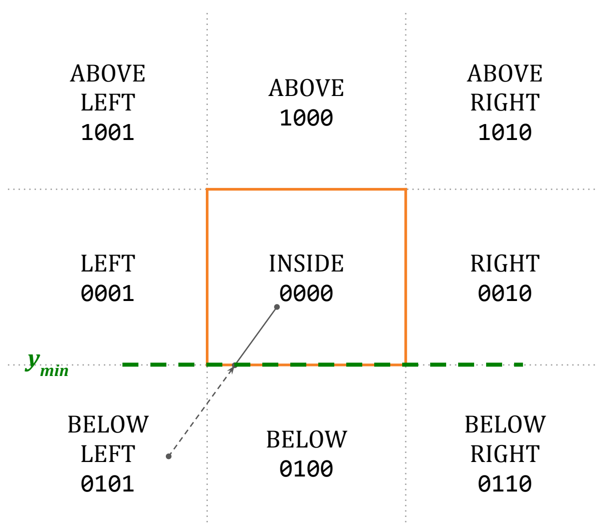
Cohen-Sutherland Line Clipping

- Another example. This line has one point INSIDE and one point BELOW LEFT.
- The point in BELOW LEFT will be manipulated.



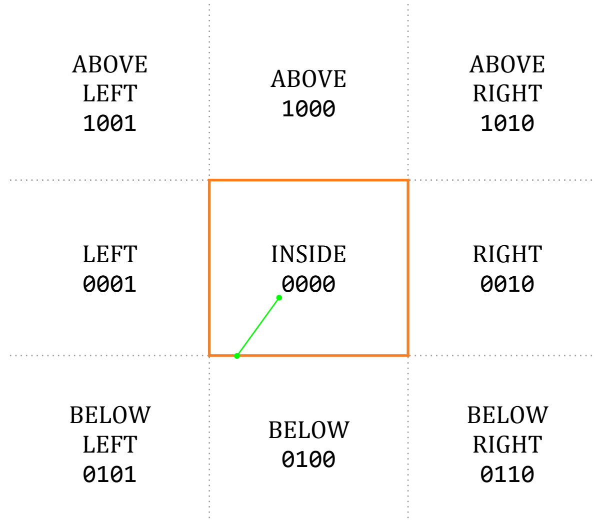
Cohen-Sutherland Line Clipping

- Another example. This line has one point INSIDE and one point BELOW LEFT.
- The point in BELOW LEFT will be manipulated.
- Since it is BELOW, it will get slid to the y_{min} line.



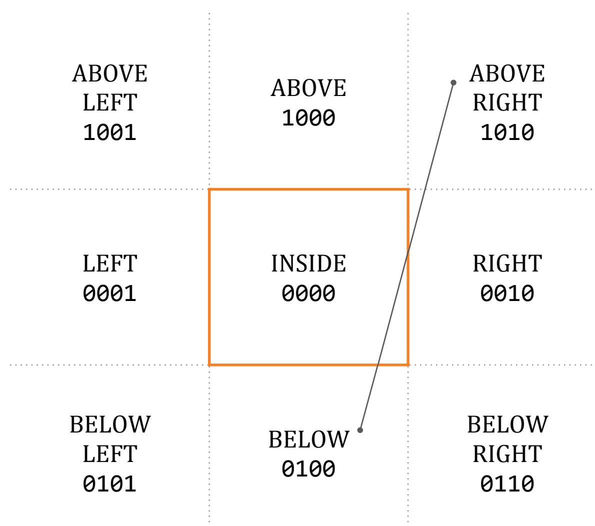
Cohen-Sutherland Line Clipping

- Another example. This line has one point INSIDE and one point BELOW LEFT.
- The point in BELOW LEFT will be manipulated.
- Since it is BELOW, it will get slid to the y_{min} line.
- When we recompute its code, both points are now INSIDE so trivial accept.



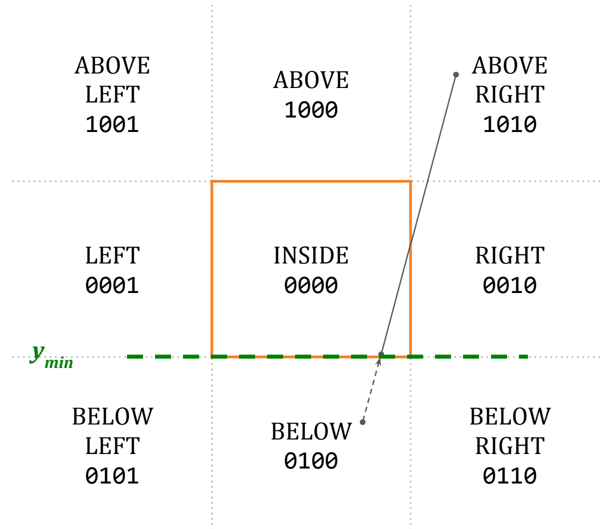
Cohen-Sutherland Line Clipping

- Final example. This line has both points outside the INSIDE region.
- The point in BELOW will be manipulated.



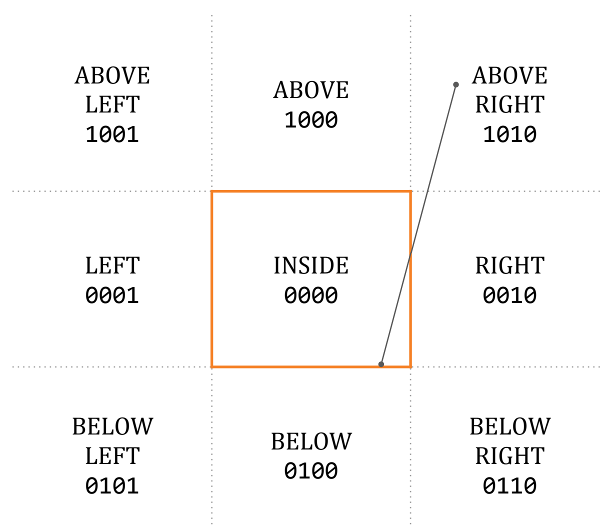
Cohen-Sutherland Line Clipping

- Final example. This line has both points outside the INSIDE region.
- The point in BELOW will be manipulated.
- Since it is BELOW, it will get slid to the y_{min} line.



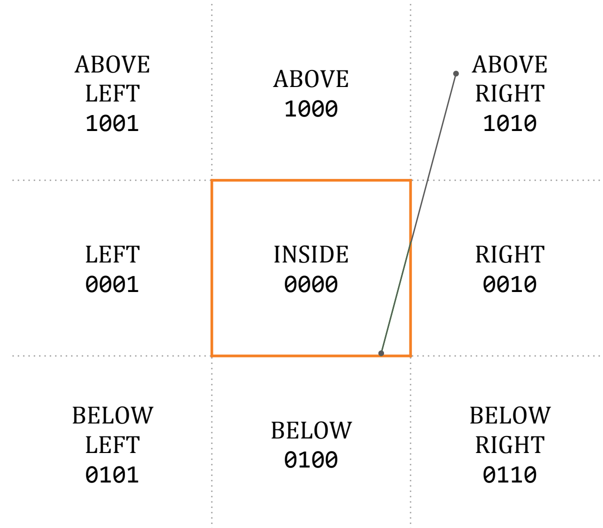
Cohen-Sutherland Line Clipping

- Final example. This line has both points outside the INSIDE region.
- The point in BELOW will be manipulated.
- Since it is BELOW, it will get slid to the y_{min} line.
- When we recompute its code, it is now INSIDE, but no easy accept or reject since the other point is still outside.



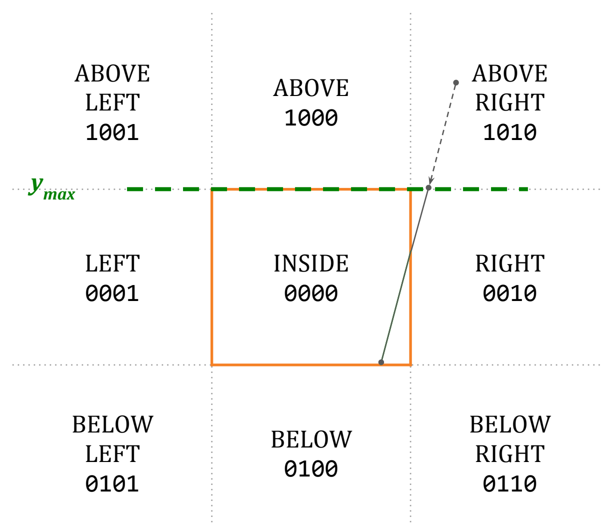
Cohen-Sutherland Line Clipping

- We next consider the point in ABOVE RIGHT.



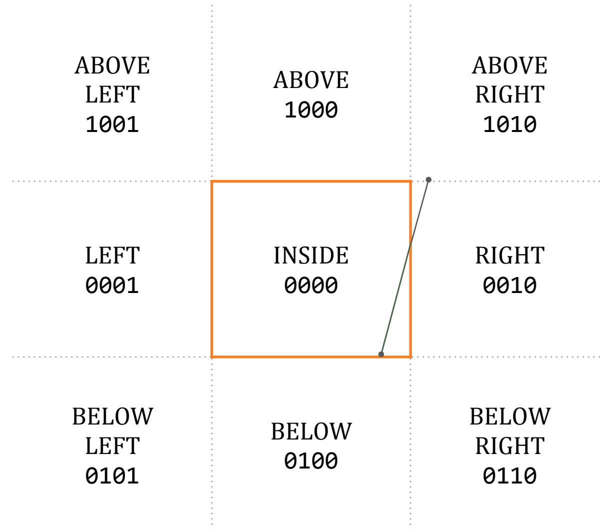
Cohen-Sutherland Line Clipping

- We next consider the point in ABOVE RIGHT.
- Since it is ABOVE, we slide it to the y_{max} line.



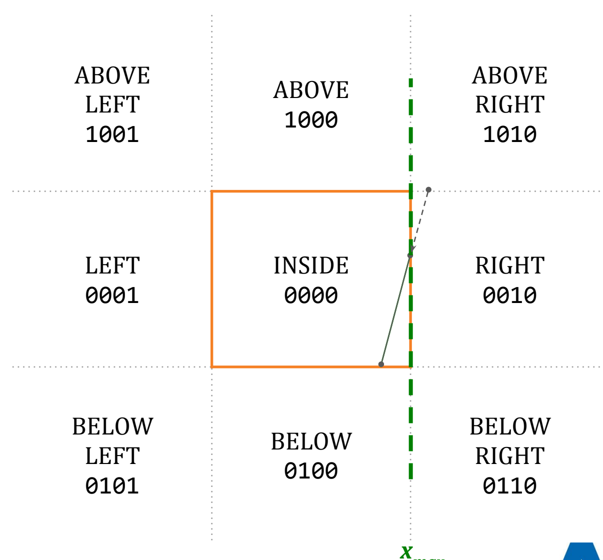
Cohen-Sutherland Line Clipping

- We next consider the point in ABOVE RIGHT.
- Since it is ABOVE, we slide it to the y_{max} line.
- Its recomputed code is RIGHT.
- There is no trivial accept or reject.
- We consider the point again.



Cohen-Sutherland Line Clipping

- We next consider the point in ABOVE RIGHT.
- Since it is ABOVE, we slide it to the y_{max} line.
- Its recomputed code is RIGHT.
- There is no trivial accept or reject.
- We consider the point again.
- Since it is RIGHT, we slide it to the x_{max} line.



Cohen-Sutherland Line Clipping

- After recomputing its code again, it is now INSIDE.
- Since both points are now INSIDE, trivial accept.

- Notice that neither of the two original end points are being used to draw the line.