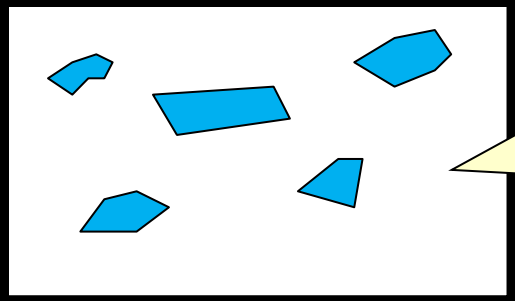
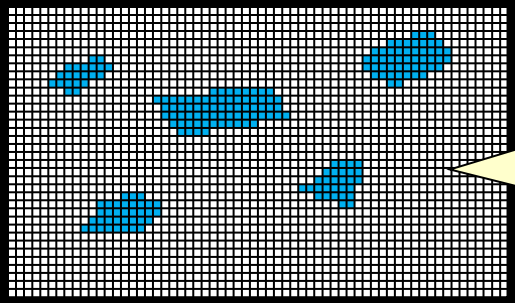
Finding Obstacle Borders

Raster Vs. Vector Maps

- Recall that large environments with few and simple obstacles take less space to store as vector than as occupancy grid:



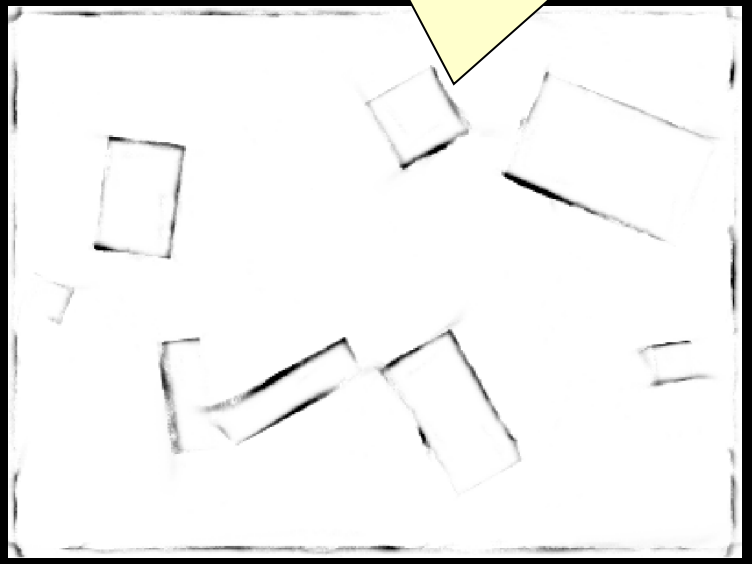
Only need to store a few vertex coordinates and edge connections.



Both “occupied” and “empty” regions take up storage space.

Our occupancy grids represent **400 x 300** grid cells (i.e., 120K).

Since each cell stores a float, it requires 120K * 4 = **480K** of storage to represent the map.

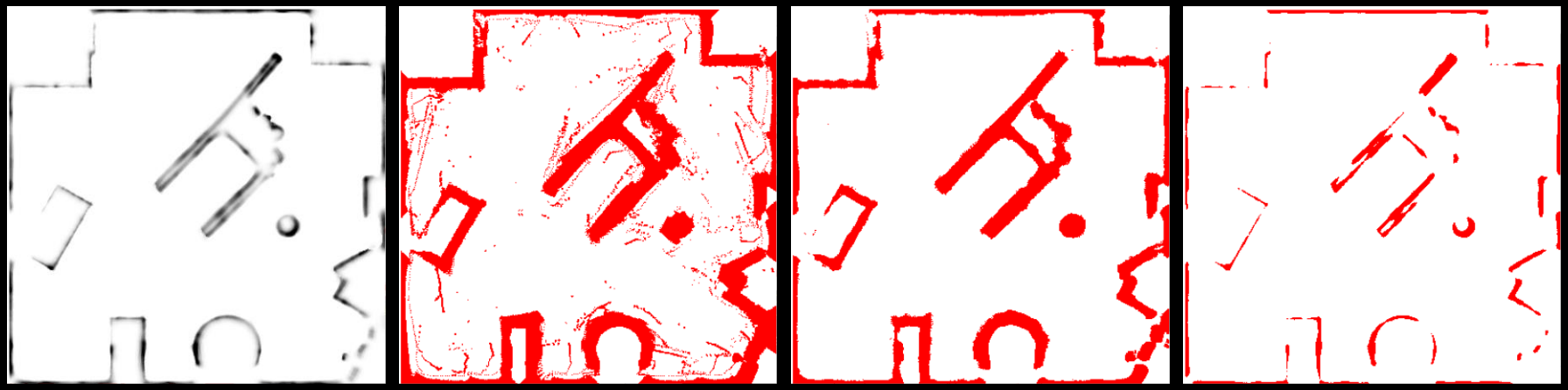


Raster Vs. Vector Maps

- It would be better to store the map as a vector
  - will take up less space
  - can represent objects as polygonal models
- Doing this will require 3 steps:
  1. Decide which grid cells are actually considered occupied
  2. Determine the borders of the obstacles
  3. Convert the borders into polygons
- In this lab period, we will concentrate only on the first two steps.

Step 1: Thresholding

- Since map cells represent the "probability" that the cell is part of an obstacle, we must decide on whether or not a particular grid cell is occupied.
- Need to have a binary grid by applying a threshold:

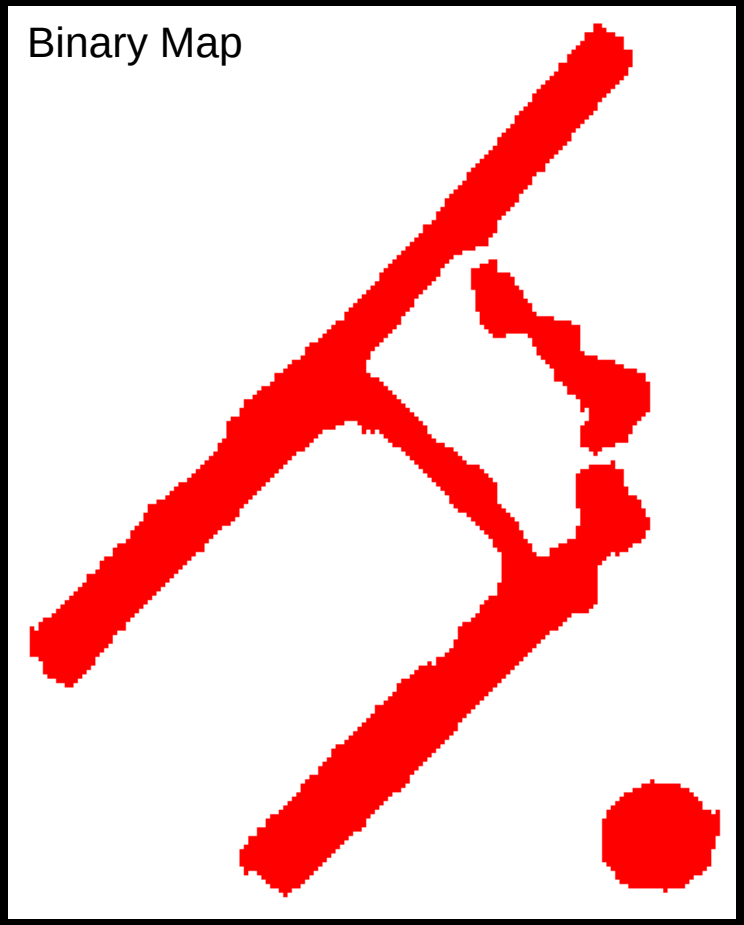


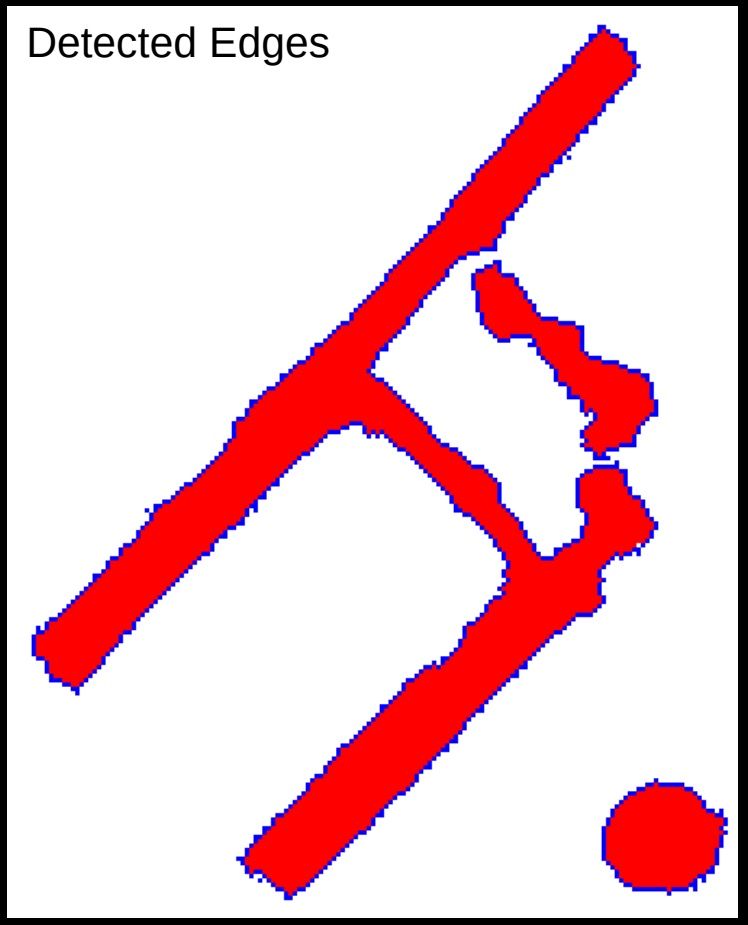
Original Grid | cells > 0 | cells > 1 | cells > 75

- We will now work with the binary grid instead of the original.
  - Grid cell values are: 0 = if no obstacle or 1 = if obstacle

Step 2: Border Detection

- We now need to process the grid to determine the borders of all obstacles.



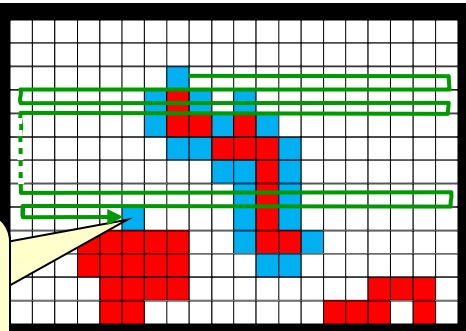


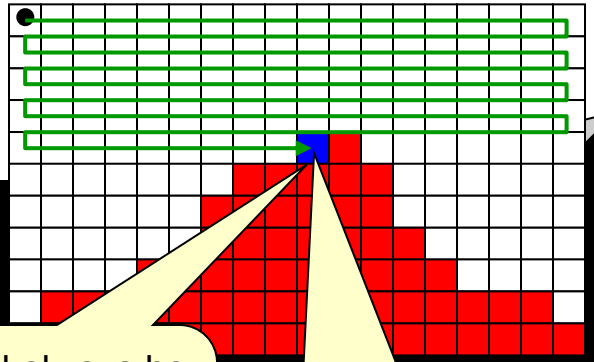
Finding the Start of Each Border

- Begin border detection by finding any point on a border. Easiest way is to check all cells starting from top to bottom going left to right until a non-zero cell is found:

This loop goes backwards!

```
FOR each Y from height-1 to 1 {
    FOR each X from 0 to width-1 {
        IF ((grid[X][Y] == 1) AND ((X == 0) OR grid[X-1][Y] == 0)))
            grid[X][Y] = 2; // i.e. BORDER
            TraceWholeBorder(grid, X, Y)
    }
}
```





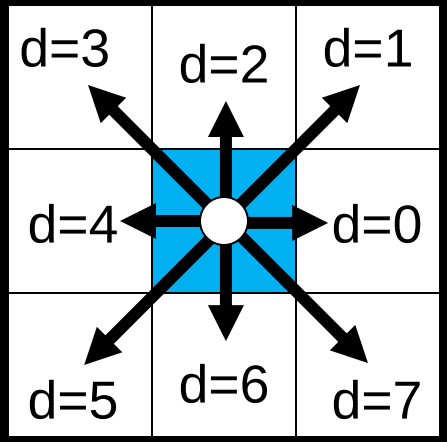
After one obstacle done, go back up to look for the start of the next obstacle to be traced.

(**X**,**Y**) will always be the top left corner of a new obstacle to be traced.

As we trace, we will set border cells to have a value of 2.

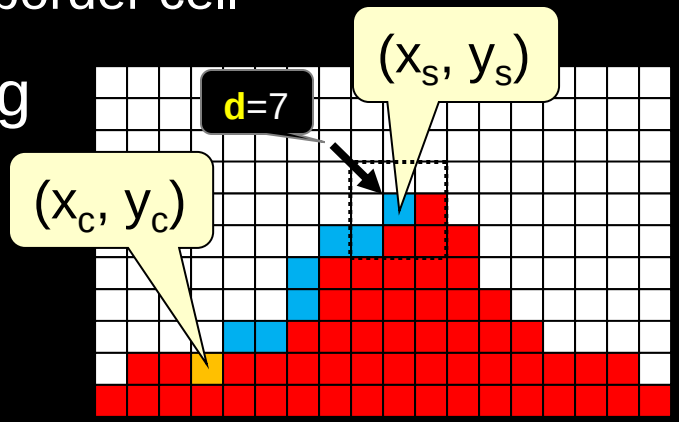
TraceWholeBorder() Algorithm

- Define direction **d** around a cell to be an integer from **0** to **7** as shown here and then define **d'** as follows:



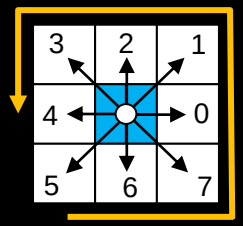

```
IF (d is even) THEN
    d' = (d+7) modulus 8
OTHERWISE
    d' = (d+6) modulus 8
```

- Border will be traced counter-clockwise (CCW)
  - **d** is direction we came in from for border cell just added
  - **d'** is direction to start looking at to find next border cell
- Begin trace with start cell (x_s, y_s) coming from direction **d** = 7, and maintain a current point (x_c, y_c) which is the last point that we added to the border.



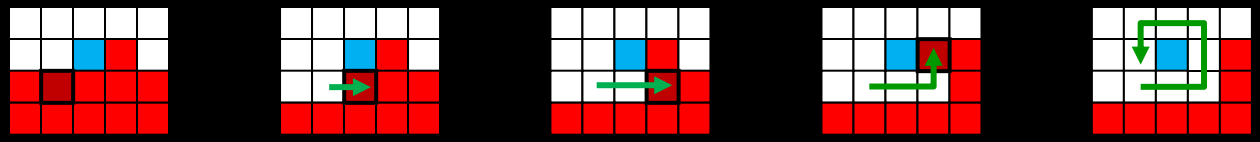
TraceWholeBorder() Algorithm

- To find the next border cell do this:
  - Starting with direction **d′**, check pixels around (x_c, y_c) in CCW order (i.e., increment **d′**) until either:
    - a non-zero (i.e., obstacle) cell is found,
    - or all 8 directions were checked and no non-zero cell was found.



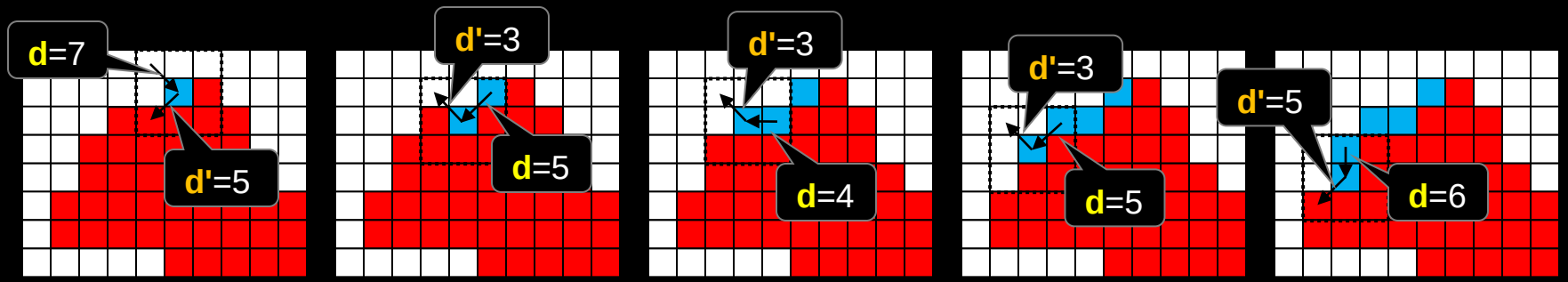
This means we found the next border cell.

- Here are some scenarios starting with **d′** = 5:



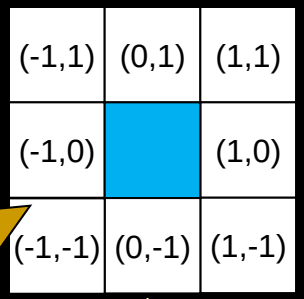
found in position 5 | found in position 6 | found in position 7 | found in position 0 | none found

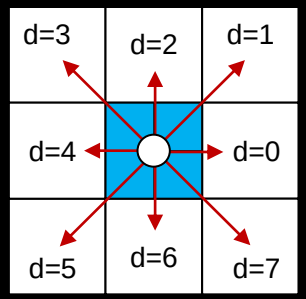
- Notice how d and d' change as the trace proceeds:



Algorithm Pseudocode

```
traceWholeBorder(xs, ys) {
    xc = xs
    yc = ys
    d = 7
    REPEAT {
        IF (d is even) THEN
            d' = (d+7) modulus 8
        OTHERWISE
            d' = (d+6) modulus 8
        found = false
        REPEAT 8 TIMES {
            tempX = xc + the xOffset in direction of d'
            tempY = yc + the yOffset in direction of d'
            IF tempX and tempY are NOT beyond the grid boundaries THEN {
                IF grid at cell (tempX, tempY) is non-zero THEN {
                    xc = tempX
                    yc = tempY
                    set grid cell at (xc, yc) to be a border (i.e., 2)
                    d = d'
                    found = true
                    break out of the inner repeat loop
                }
            }
            d' = (d'+1) modulus 8;
        }
        IF not found THEN {
            set grid cell at (xc, yc) to 0
            done tracing … quit the function
        }
        IF (xc, yc) is the same as (xs, ys) THEN
            done tracing … quit the function
    }
}
```



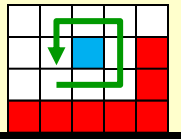


(xOffset,yOffset) values for each direction

Happens when (xc, yc) is on the grid boundaries. Need to check for this otherwise we can get an ArrayIndexOutOfBoundsException

DO NOT use the **isInsideGrid()** function.

Must be a single point since tracing all 8 directions did not find a non-white cell. So, erase it.



We finished tracing when we reach the start again.

Start the Lab …