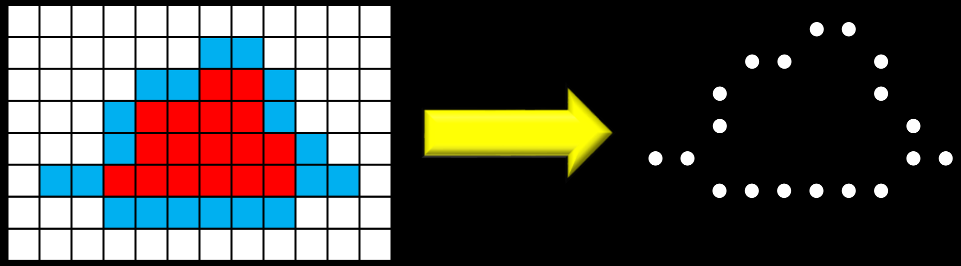
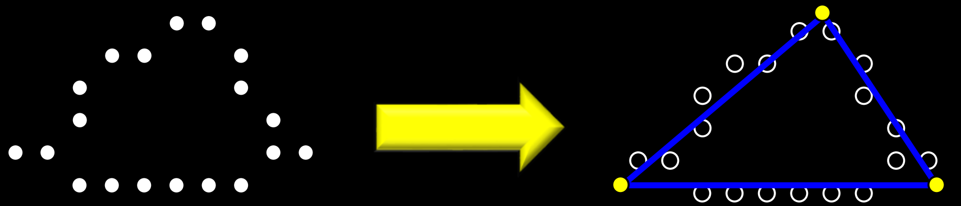
Converting to Vector Maps

Converting To Vector

- Once we have detected object borders within the occupancy grid, we can convert to polygons.
  - Consider each obstacle separately.
  - When tracing a border, build up a list of the border points for that obstacle in sequential order … assume each grid cell is a point.



  - Then run a line-fitting algorithm



Converting To Vector

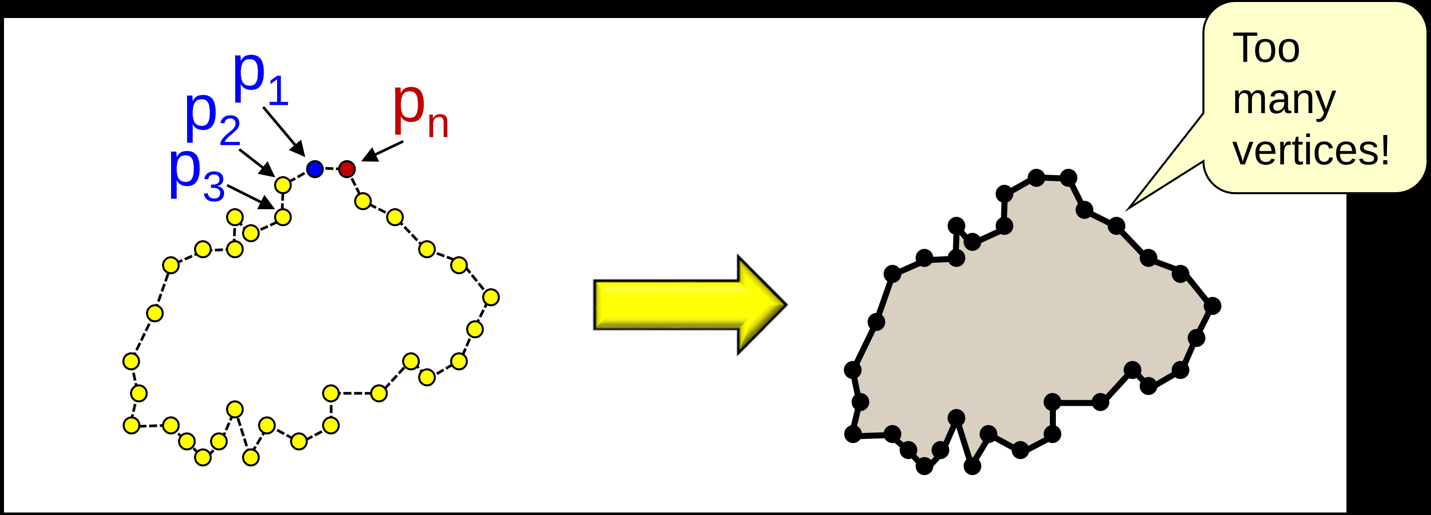
- Key issues:
  - – How do we group points into line segments ?
  - – How can we detect and eliminate noisy data ?

- There are a variety of common techniques:
  - – Split & Merge
  - – Incremental }
  - – Line Regression
  - – RANSAC (Random Sample Consensus)
  - – Hough-Transform
  - – EM (Expectation-Maximization)

Simplest and popular …
we will use this one.

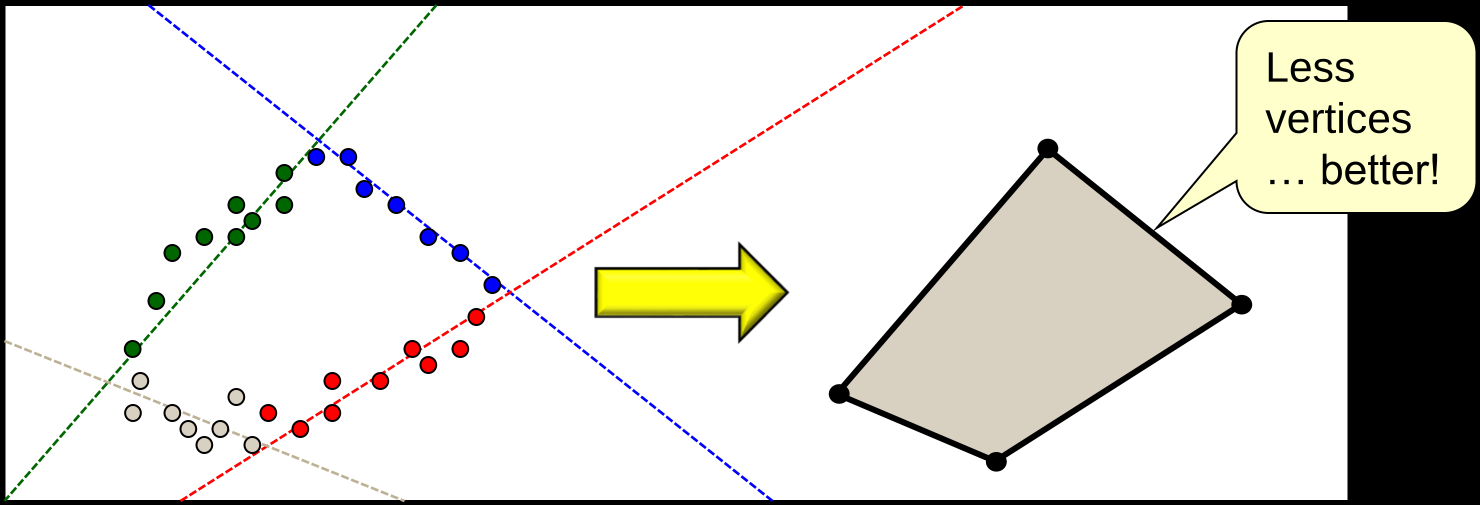
The Main Idea

- Consider a set of border points:
  - $P = \{p_1, p_2, p_3, \ldots, p_n\}$ where $p_i = (x_i, y_i), 1 \leq i \leq n$
  - As we do a border-tracing algorithm, these points will be coming in a counter-clockwise order
- We can simply connect all points in order, but this is too many points and assumes that all points are valid.



The Main Idea

- It is better to try and “fit” lines to the data:



Less vertices … better!

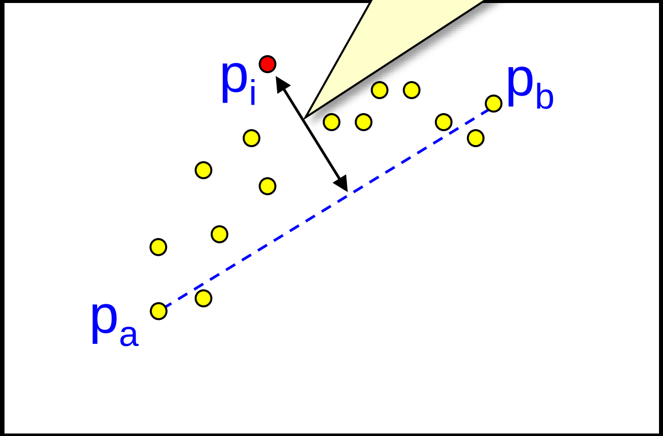
- But how do we know which points fit to a line ?
  - Assume consecutive points lie on the same line unless they are too far away from the line.

The Main Idea

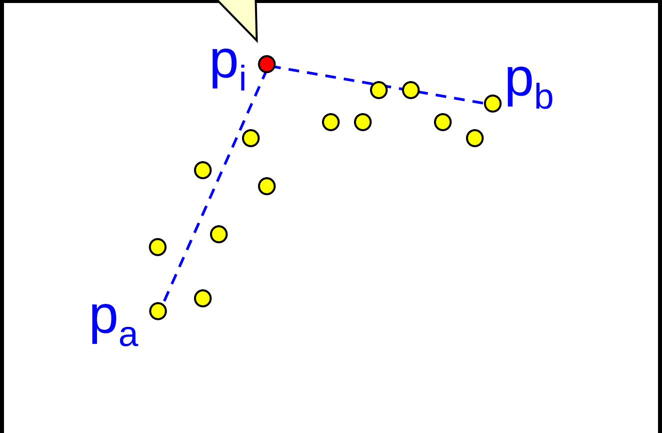
- Let p_i be point of maximum distance from line L = $\overline{p_a p_b}$

Distance from $\mathbf{p_i}$ to **L** is:

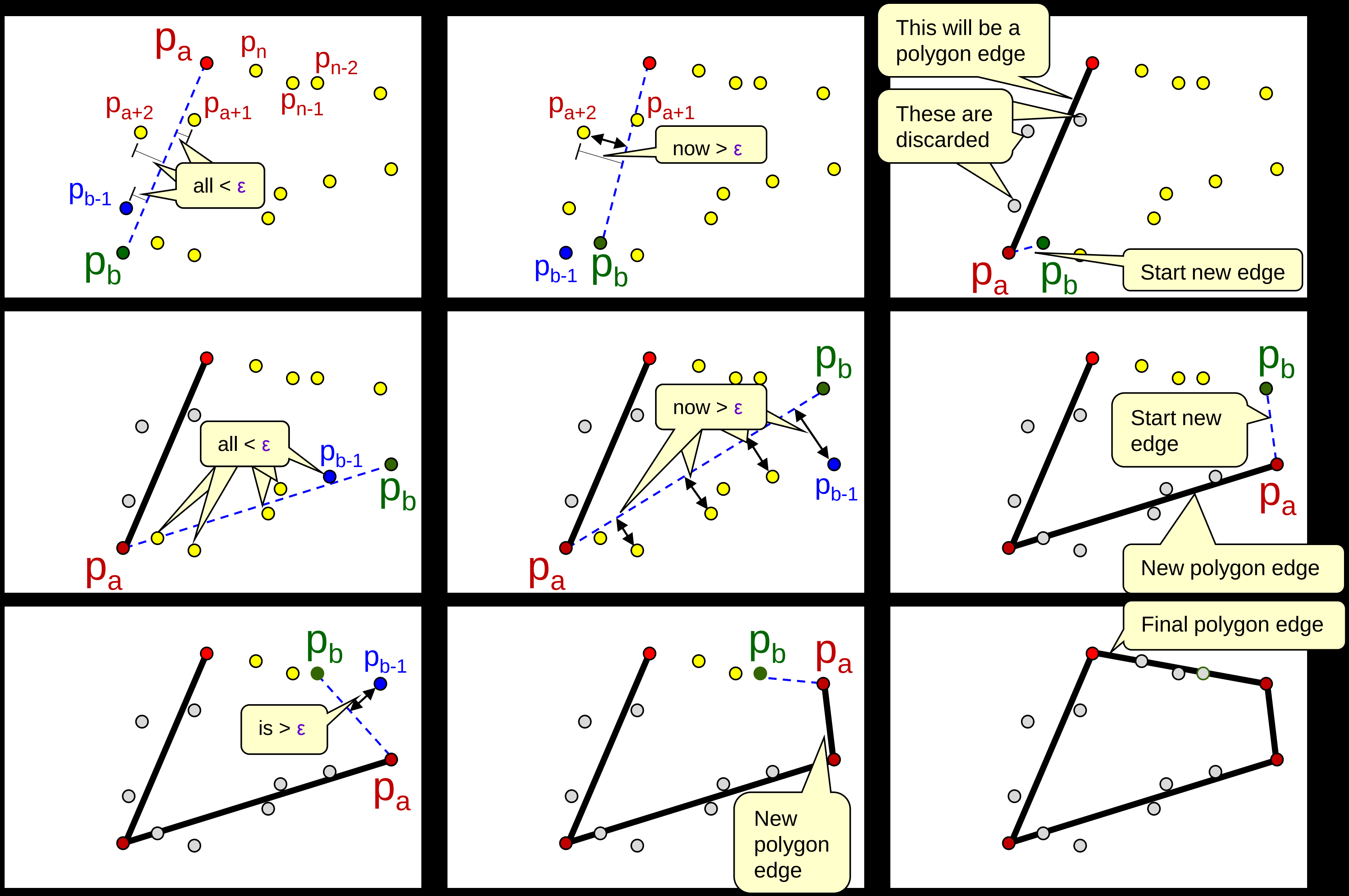
$$\frac{|(x_b-x_a)(y_a-y_i)-(x_a-x_i)(y_b-y_a)|}{\sqrt{(x_b-x_a)^2 + (y_b-y_a)^2}}$$



If $\mathbf{p_i}$ is too far away, then this data must represent more than one line (i.e., two distinct polygon edges).



The Incremental Algorithm



The Pseudocode

- Here is the algorithm for point set p_1, p_2, p_3, …, p_n:

```
Set polygon to be a new obstacle with no vertices
First, make sure the point set has at least 3 points, otherwise quit
Set $a = 1$, and set $p_a$ to be the polygon’s first vertex
FOR index $b = 2$ to $n$ DO {
    FOR index $i = a+1$ to $b-1$ DO {
        IF point $p_i$ is too far from line $p_a p_b$ THEN {
            Add $p_{b-1}$ as the next vertex of the polygon unless it is the same as the last vertex added.
            Set $a = b-1$.
        }
    }
}
```

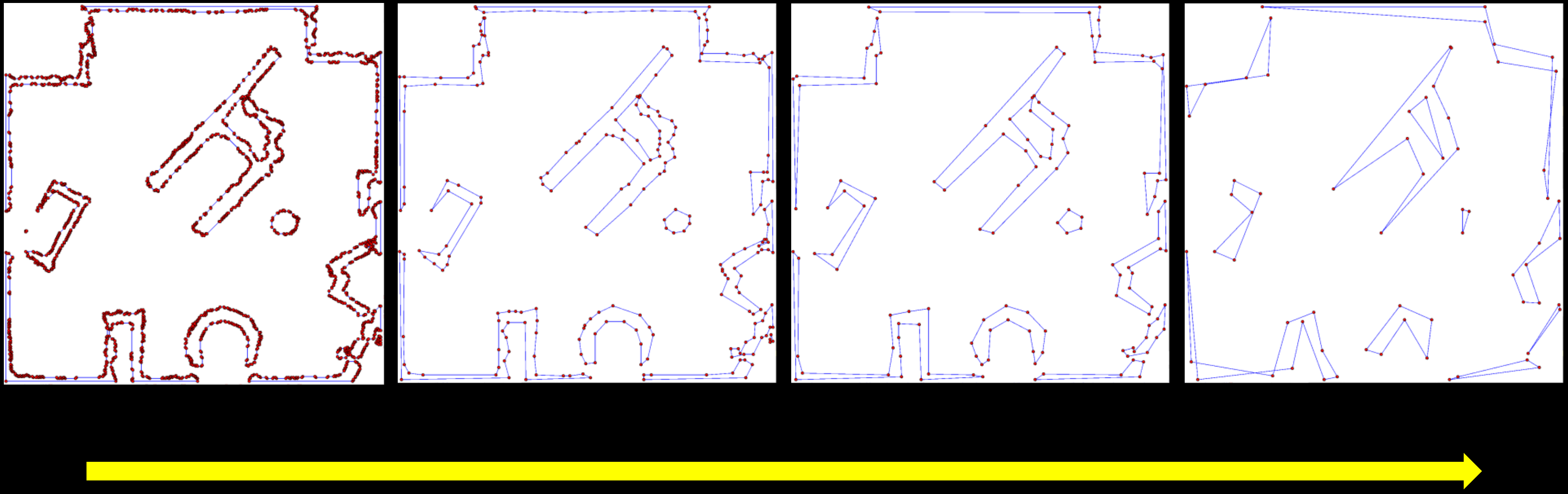
Note that the pseudocode has points with indices **1** to **n** but our **Obstacle** indices go from **0** to **n-1**.

Too far means greater than some LINE_TOLERANCE (i.e., denoted as ε in the previous slide)

- Once the polygon is created, we should check to make sure it has at least three vertices. If not, we discard it since it is not a proper polygon.

The Incremental Algorithm

- By increasing the line-fit tolerance, the number of polygon vertices decreases:



Increased Line-Fit Tolerance (i.e., Error Threshold)

Start the Lab …