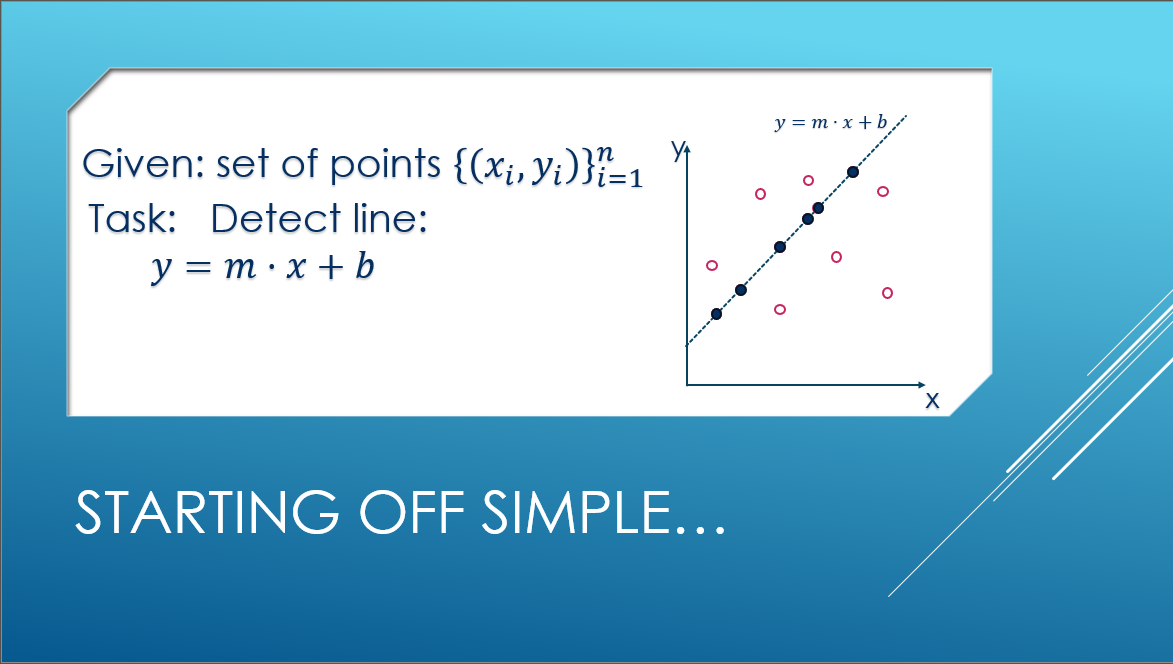
**Report**

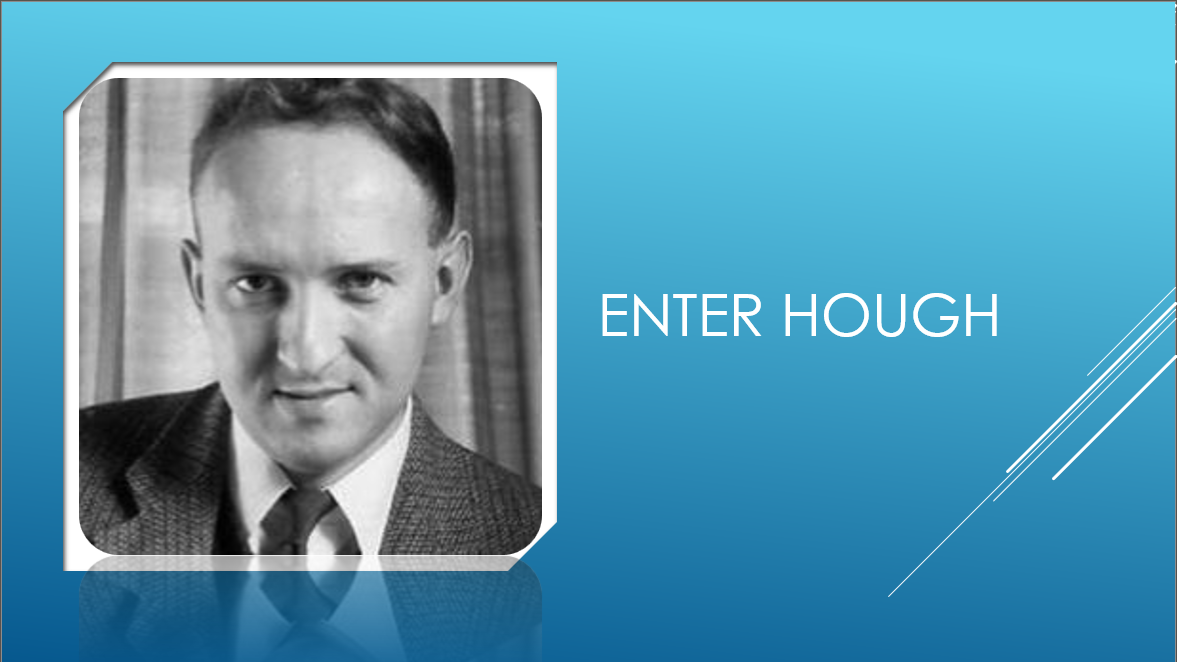
**Introduction**

Hough transform is a unique way to cultivate a dual representation of lines and points interchangeably within two spaces, the image space and the parameter space.

In this report we cover the basic idea of this duality as mentioned in class. We show our implementation of the generic accumulator matrix method introduced by Hough in 1962. We discuss the correctness of the algorithm and presents our two claims, that we can find a granularity of the discretization to satisfy completeness and soundness which we define later. We then go on to talk about the transition from a standard image to a set of edge points on which we can use the previous algorithm presented. This transition is being done by using the algorithm of John F. Canny, which we also implemented.

**Part 1**



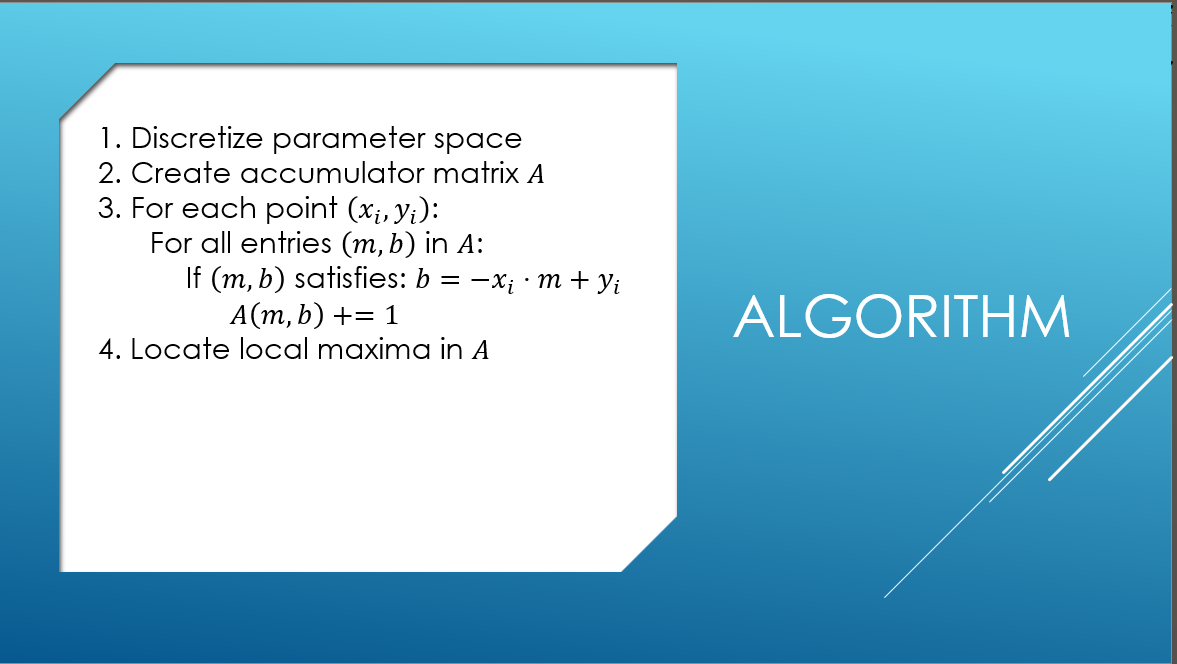


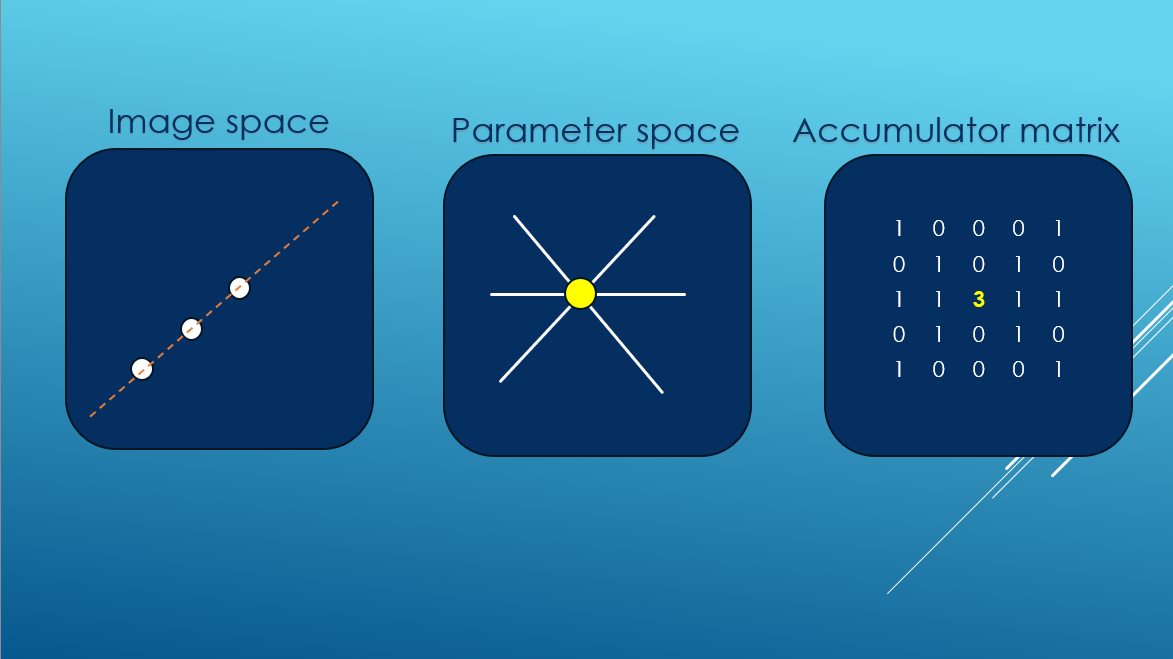
תמונה שמכילה טקסט, תרשים, קו, צילום מסך

התיאור נוצר באופן אוטומטיתמונה שמכילה טקסט, צילום מסך, תרשים, קו

התיאור נוצר באופן אוטומטי

As thought in class, looking at the parameter space we can that a line that passes through some points will be manifested as an intersection point between the lines corresponding to these points in the parameter space.



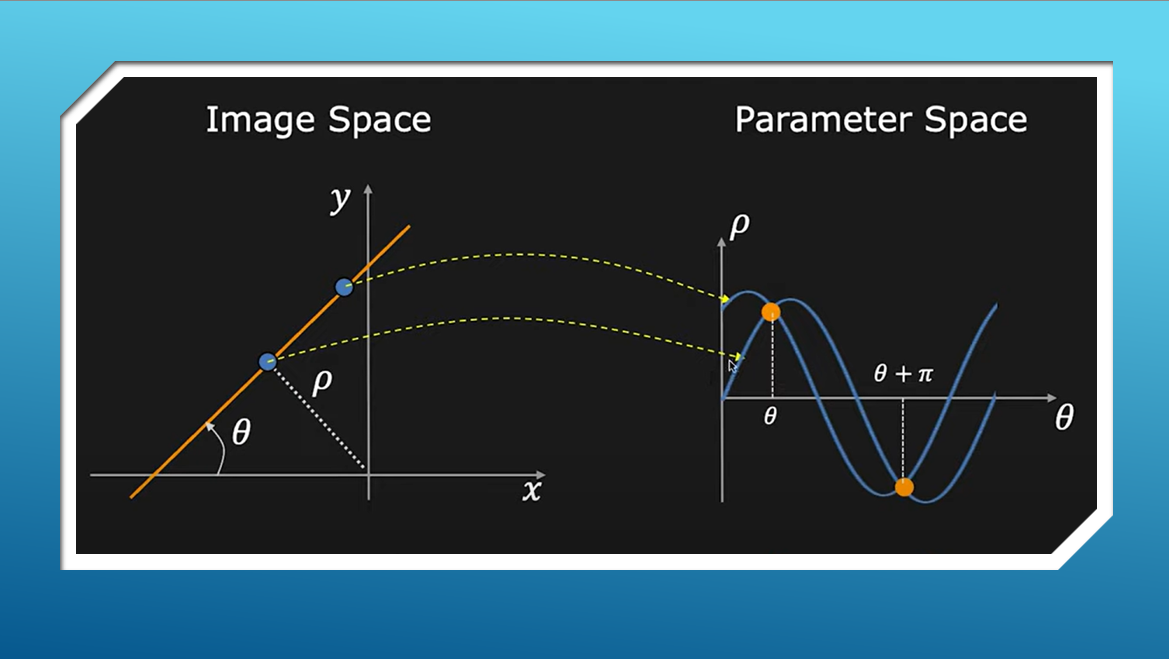


An illustration of the way that the matrix should be filled

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי

A problem arises when devising an implementation for this idea, since the parameter for the slope is not bounded, making it impossible to be represented fully in a discrete finite matrix.

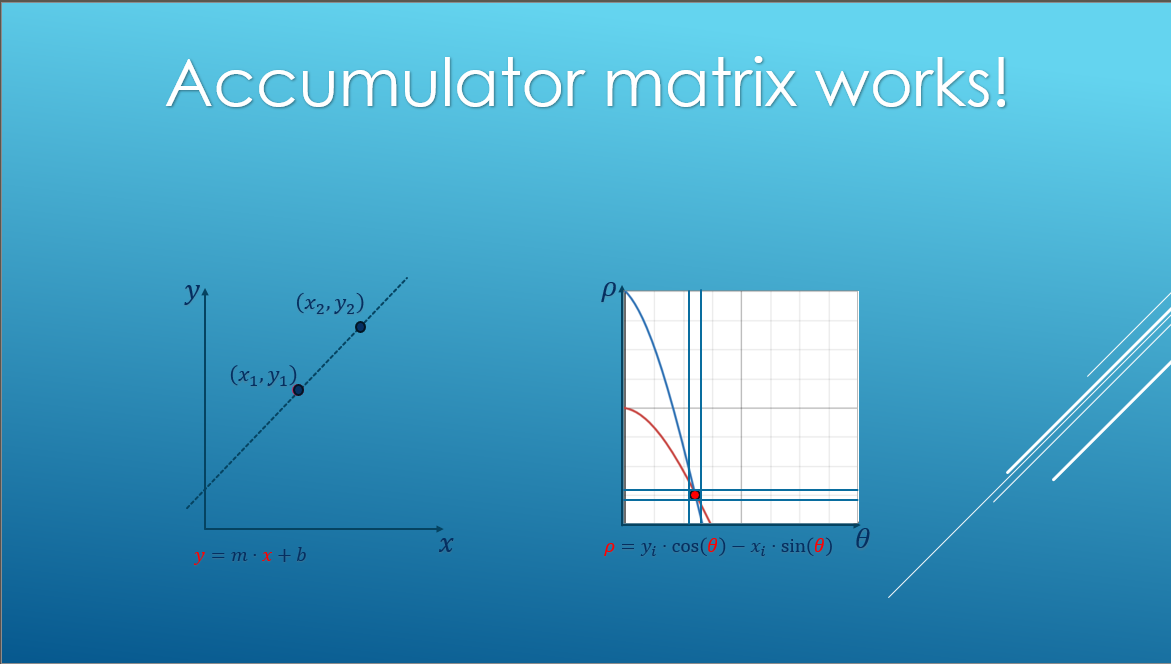


The representation with as the offset from and the angle between the line and axis represents a unique line now with bounded parameters. With and where .

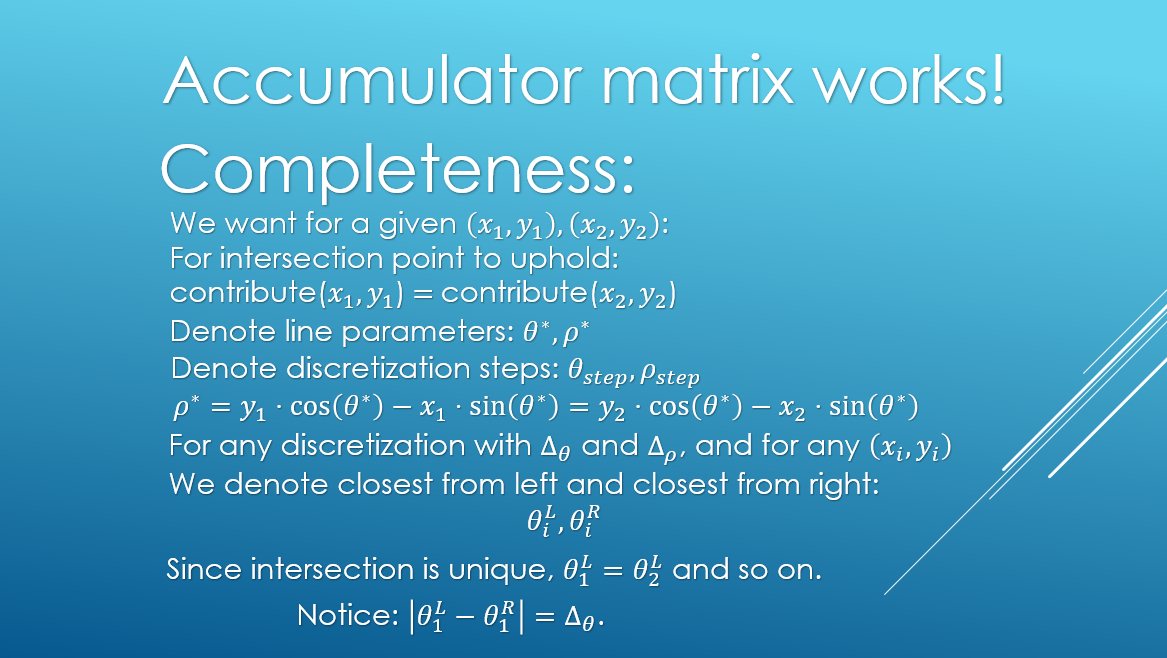
**Part 2**

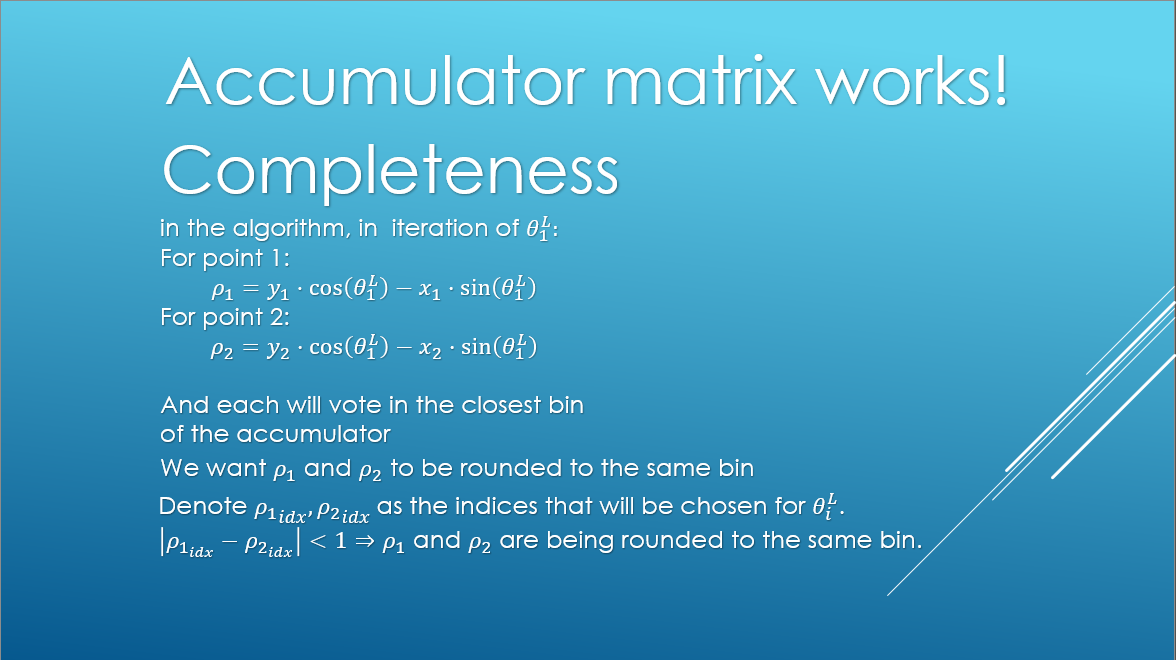
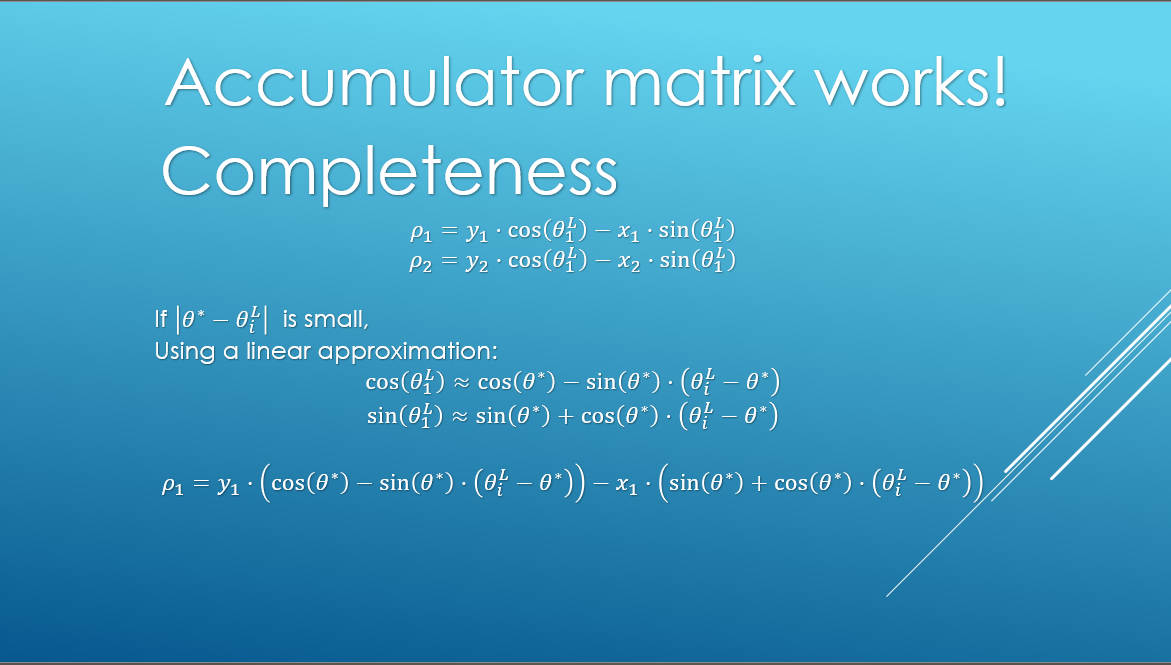
We discretize our space, what guarantees that we catch every intersection? Secondly What guarantees that irrelevant points don’t vote in cells of the accumulator inadvertently?

**Completeness:**

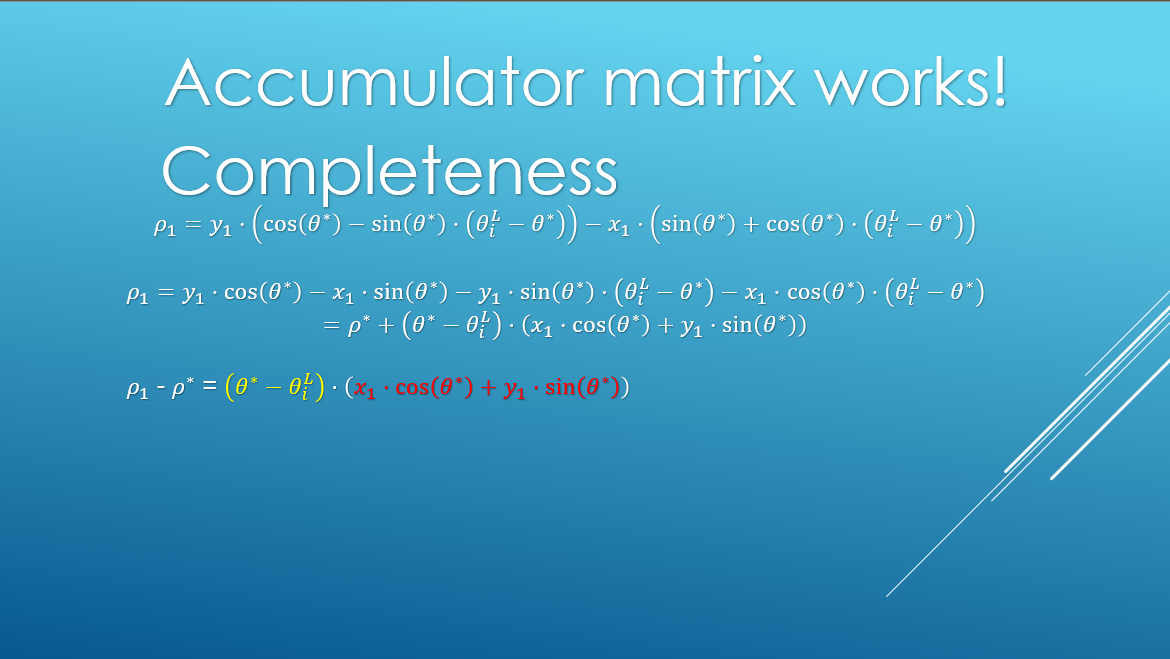


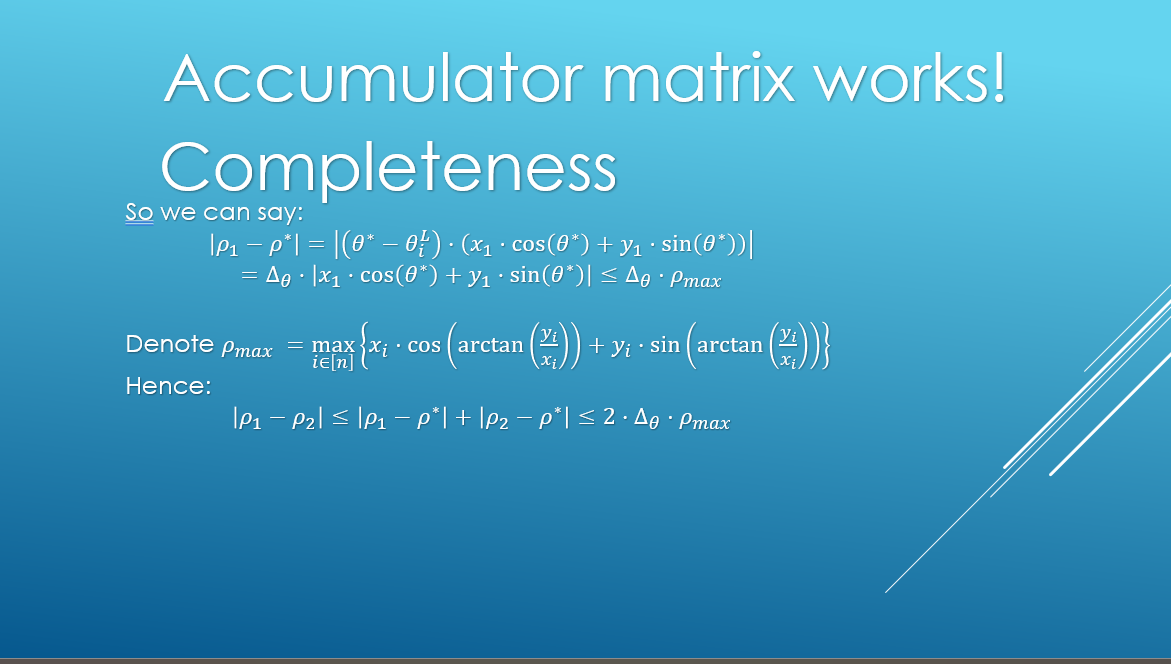
We want to show that we can make a requirement that will necessarily make the intersection point be caught, meaning that when different points on the same line will vote in matrix cells, the values that will be chose won't be too far apart and scattered, rather they will be closed in two adjacent cells in the matrix.

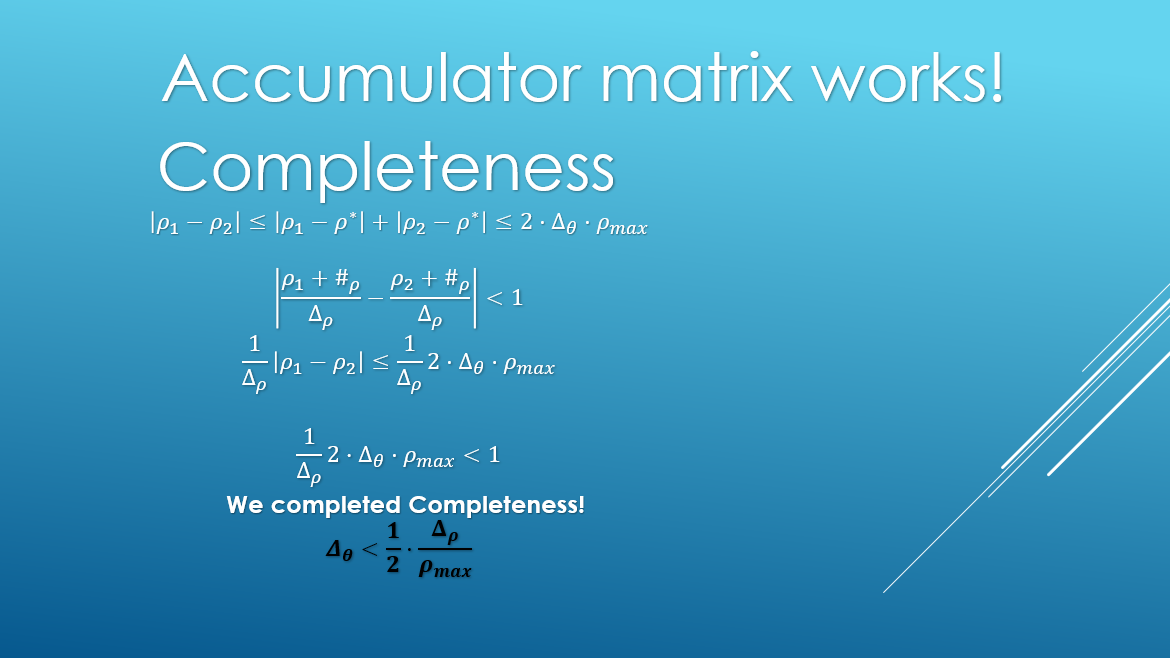
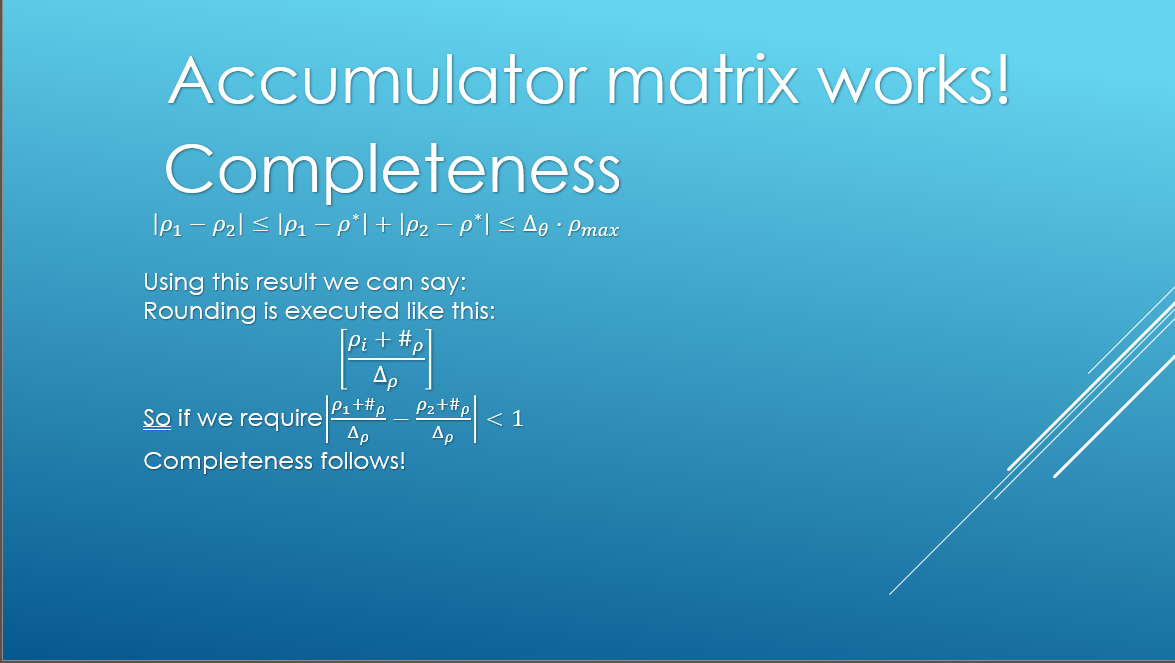


תמונה שמכילה טקסט, צילום מסך, גופן, עיצוב

התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט, צילום מסך, גופן, עיצוב

התיאור נוצר באופן אוטומטי



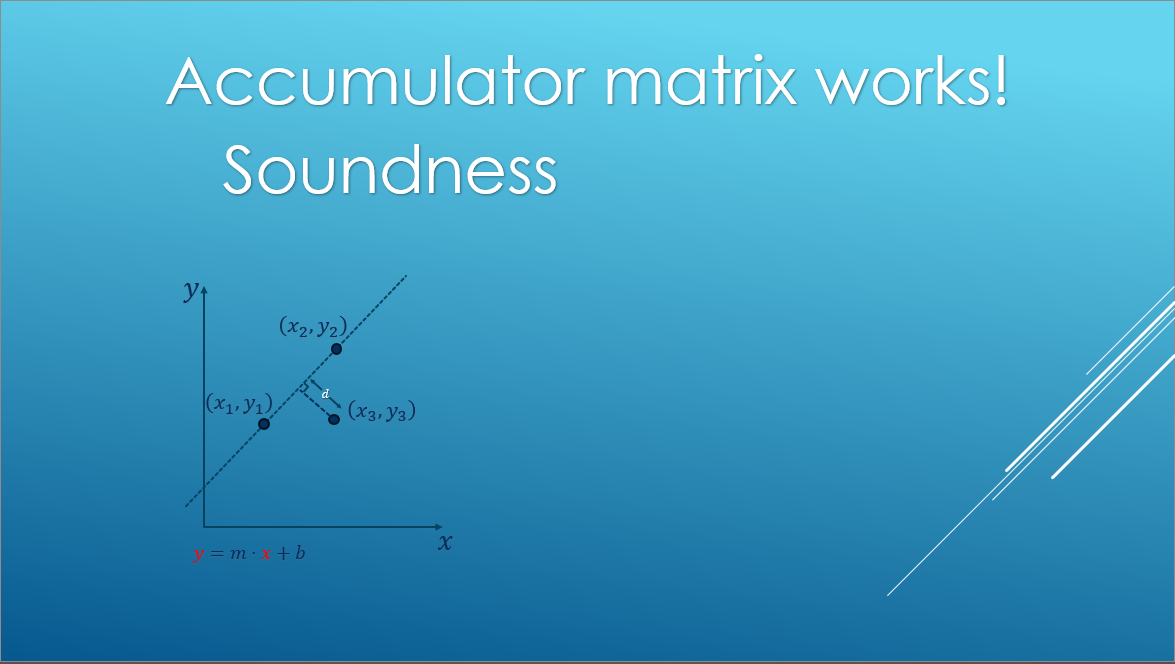
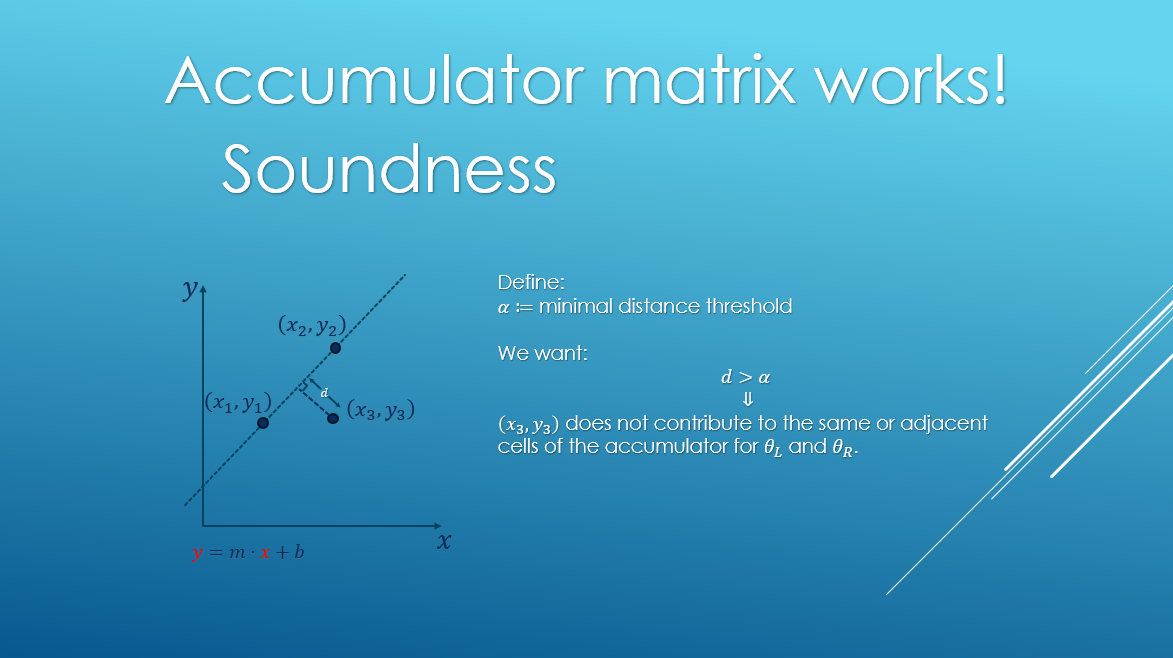
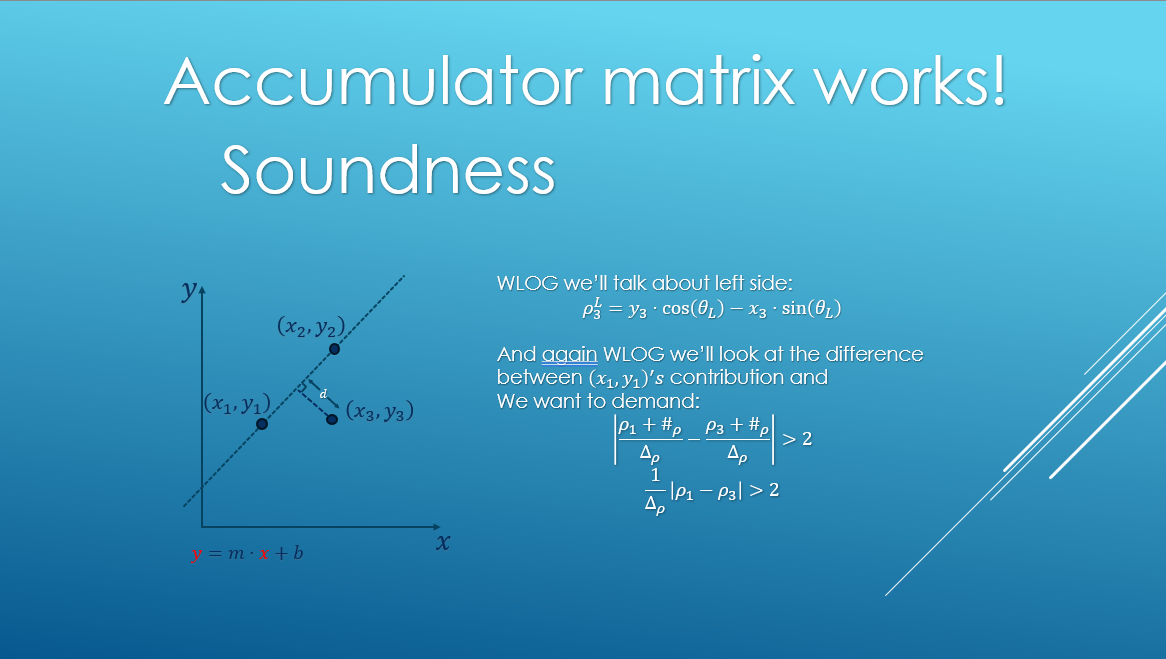
What we showed is that given a step in : , satisfying , for any two points that are on the same line, for the closest in the discretization to the real of the intersection point in the parameter space, for both points the index that will be chosen in the algorithm will be at most 1 cell apart in the accumulator matrix. Satisfying this for any two points on a line necessitates that for any points in the same line the claim holds for any pair, meaning that for all points, there are at most 2 adjacent cells for that they vote to when the algorithm select or (symmetric cases).

That way it is guaranteed that the intersection point will indeed be 'caught' by a cell or two in the matrix.

**soundness**

The next thing we want to show is that we can make a requirement that makes sure that unrelated points won't contribute to a cell they do not belong to, i.e. a cell of parameters of a line it does not resides in.

For that we define as the minimal distance that make a point far from a line by being considered as not belong to.

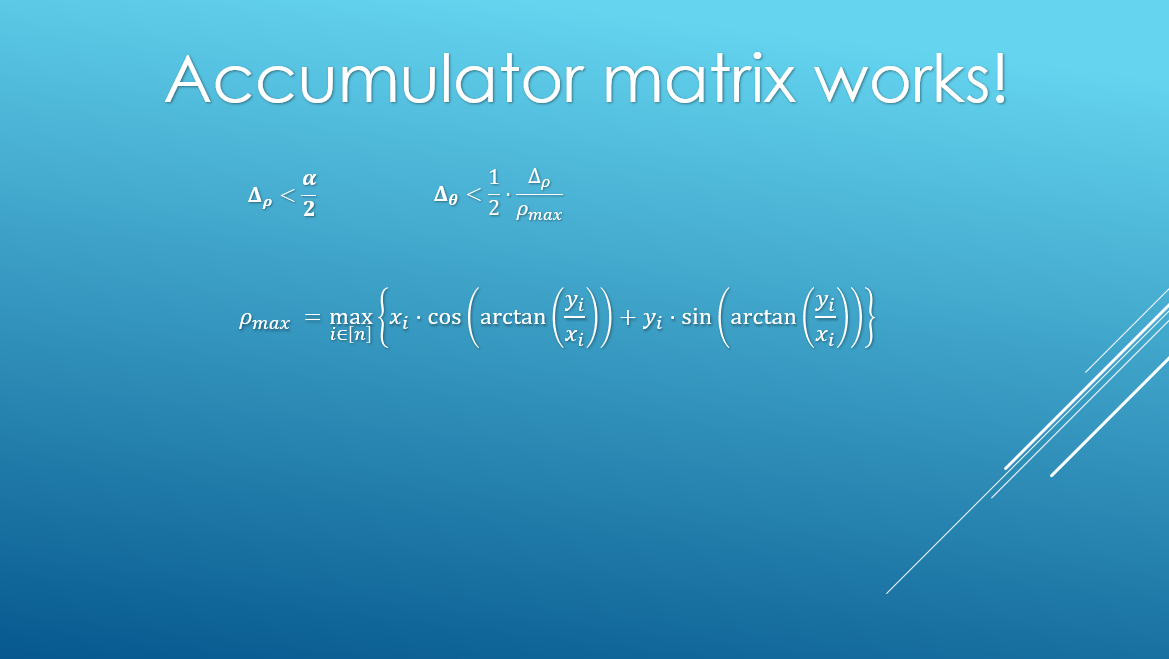
  

תמונה שמכילה טקסט, גופן, תרשים, עיצוב

התיאור נוצר באופן אוטומטי תמונה שמכילה טקסט, צילום מסך, גופן, תרשים

התיאור נוצר באופן אוטומטי

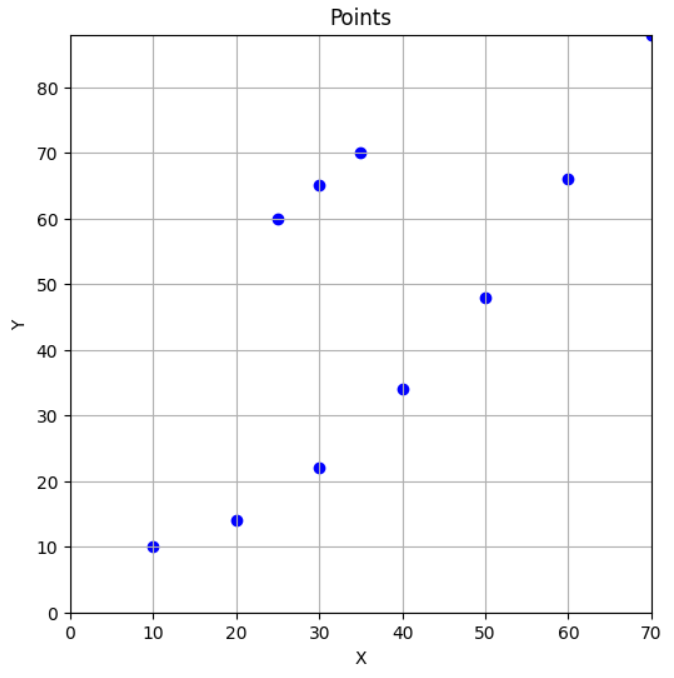
What we showed is that if choosing , a point distant more than from a line will not contribute in a cell adjacent to where any two points residing on will contribute to.



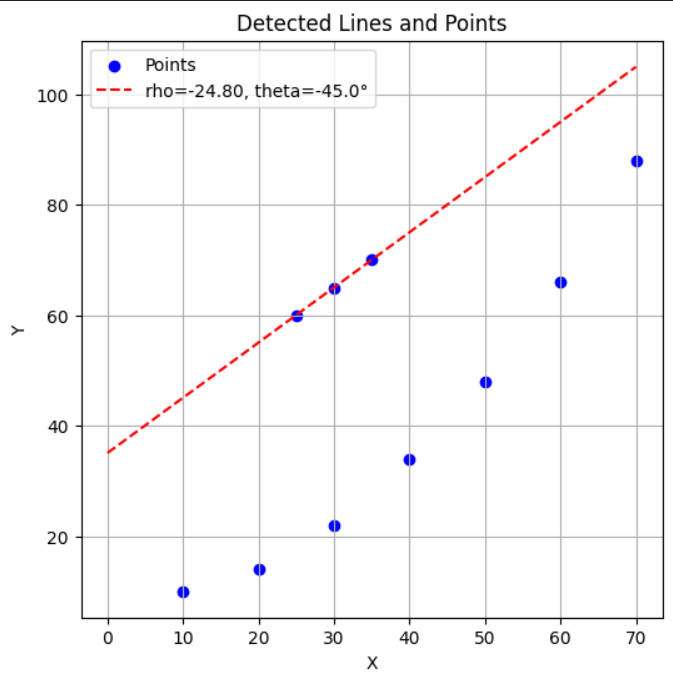
**Part 3**

In this part we show the demo of the code we implemented. The code itself is in the file *"demo.ipynb"*.

We use a sample of points to demonstrate the algorithm. we chose and tuned accordingly.

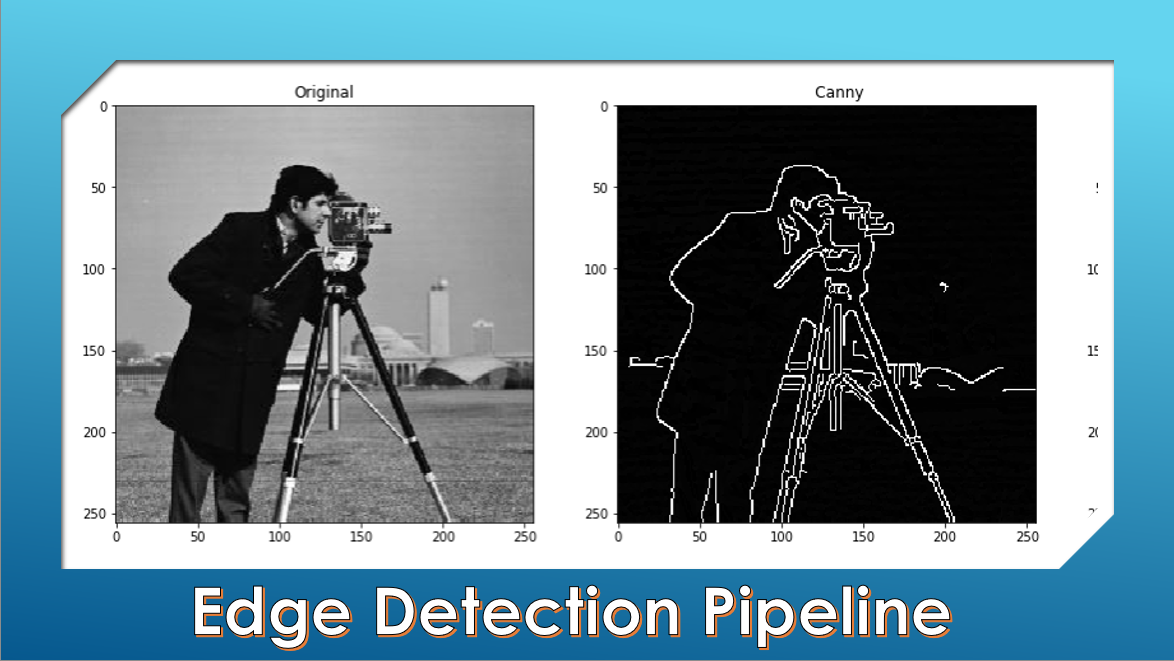


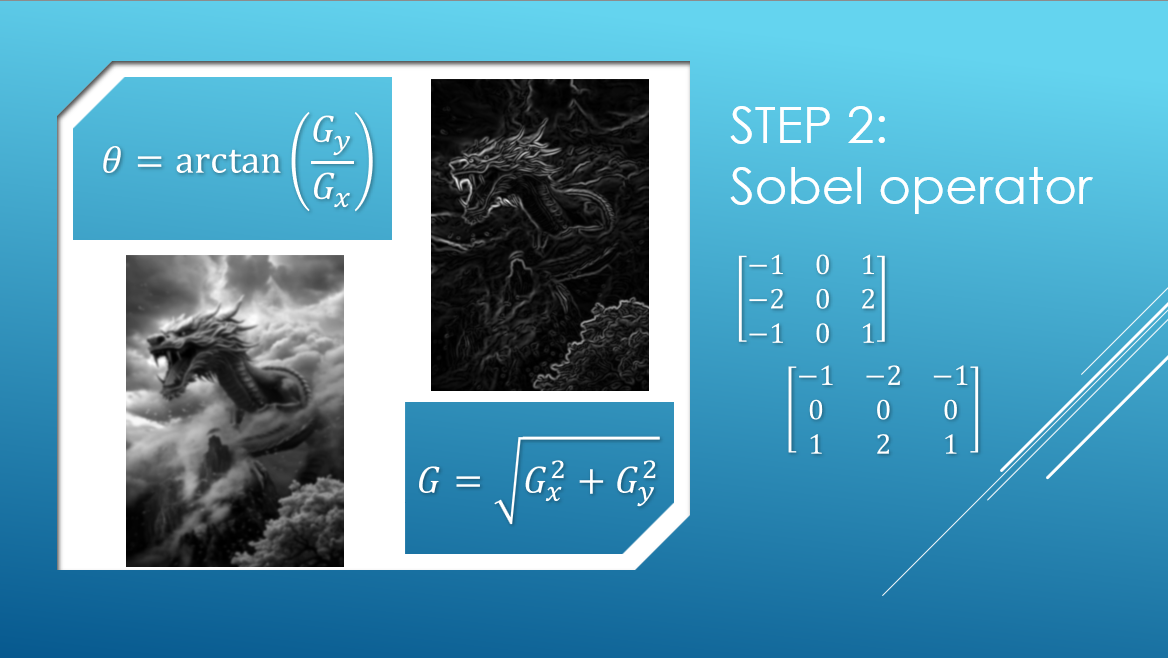
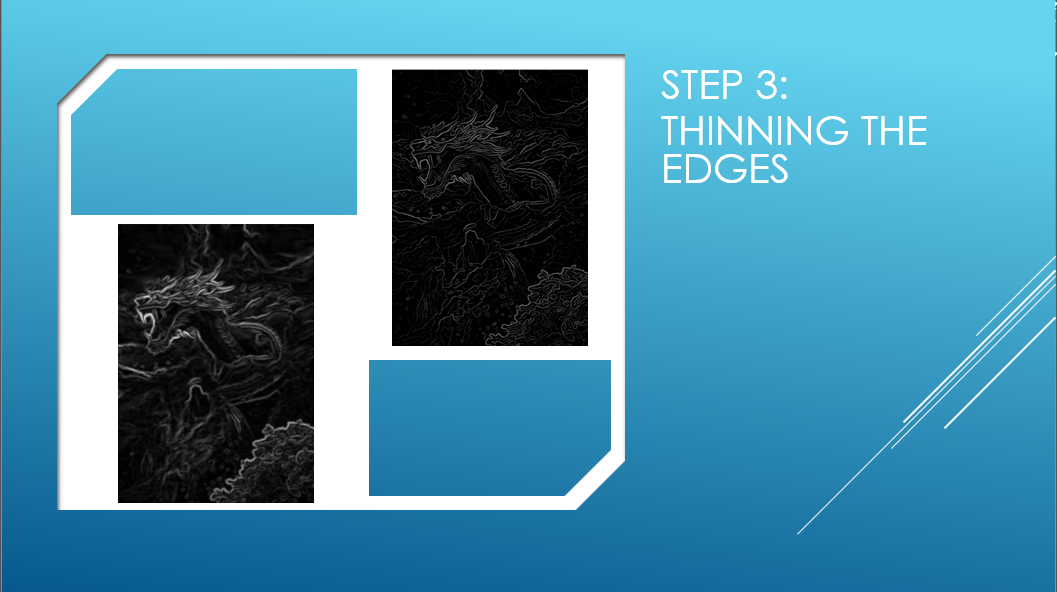


****

It can be seen that even though there is a sub-set of more than 3 points that construct a curve that can be interpreted as a line, due to our parameters tuning there is a predecence to the straighter line with less points.

**Part 4**

  תמונה שמכילה צילום מסך, טקסט, מלבן, עיצוב

התיאור נוצר באופן אוטומטי



Explanation: Or:

**Part 5**

Here we show our demo using a 'real' picture.