Triangle_finder

Approach

A) Create a binary mask separating foreground (shape, lines) from the background.

B) Segment the mask into separate shapes. See Explanation #0.

   A) Assume pixel (0,0) is background. Inpaint it as background_shape (not to be confused as background).

   B) Searching for remaining shapes, starting from (0,0), traversing the x-dim and then y-dim, to find a foreground pixel (“first pixel of shape”). Note “first pixel of shape” must be a vertex of the shape, see explanation Explanation #1.

      Then, at that foreground pixel:

      A) If it is foreground, then inpaint it as the *i*th shape. While inpainting keep track of the pixel that is farthest from the “first pixel of shape”. Note this pixel must also be a vertex, see explanation Explanation #2.

      B) If it is background, search its 3x3 neighborhood for a shape, and inpaint it as a that shape. This step fills in the shape.

C) Find the third vertex, by looking at every pixel of the shape, and returning the one with maximum distance to the line formed by the first two pixels. See also Explanation #3 and Explanation #4.

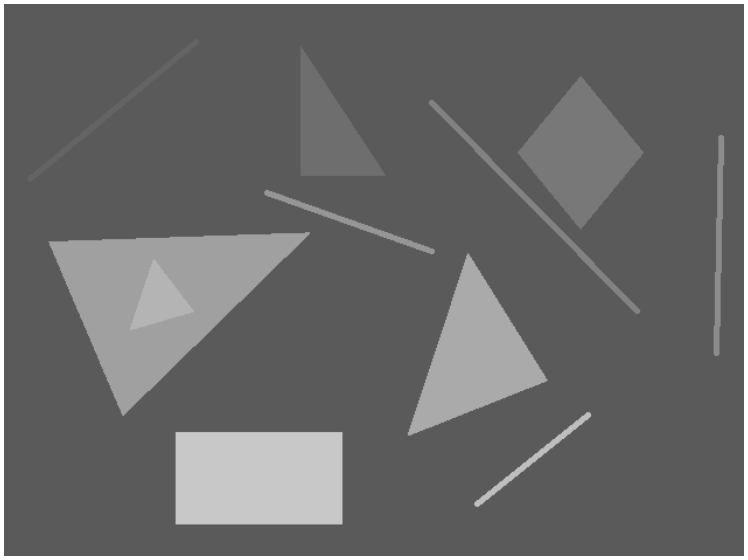
D) The three vertices form three edges. For each edge ensure that one side is filled and ensure that the other side is not filled, if this condition holds true for all edges, then the shape is a triangle. See Explanation #5.

E) Calculate analytics of the triangle. Area is half the magnitude of the cross product of any two vectors. Type is determined by measuring edge length and comparing it within a tolerance.

Explanations

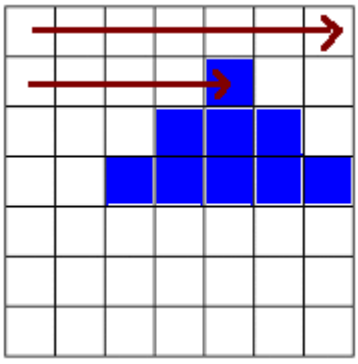
Explanation #0

The goal of segmentation is to assign a different pixel value to each shape, and infill it completely. This segmentation still respects if a triangle is inside another triangle. The background is assigned its own value. This infilling is critical for determining if the shape is a triangle (see explanation #5). In image below (result of segmentation), each shape is a slightly different shade of grey reflecting that they all have different pixel values.



Explanation #1

If we start the search at the origin and traverse the x dimension, incrementing in the y dimension after the x is done, then in theory the first pixel of the shape we will encounter will be its vertex. This might sometimes not be the case if the image is low-resolution and the shape's bottom-most edge has a slight positive slope. But the images had good enough resolution that this problem was not encountered.



Explanation #2

Given a vertex of a polygon (or line), the point in that polygon farthest from that vertex should also be vertex.

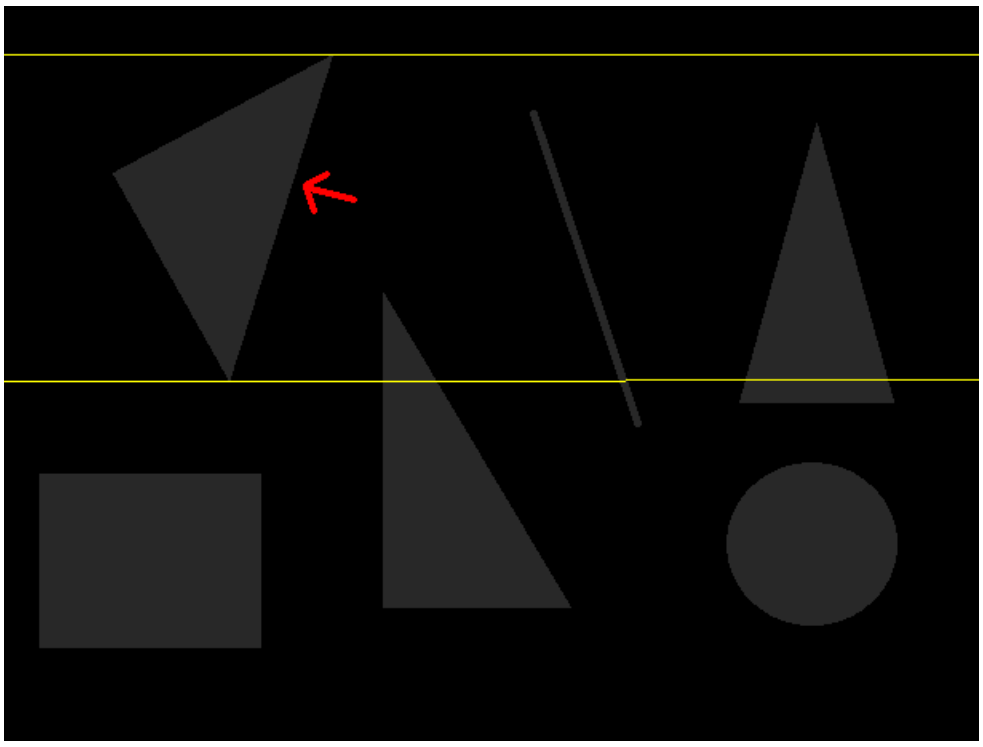
Explanation #3

Given two vertices of a polygon, they make a line. The point on that polygon farthest from that line is also a vertex of the polygon. This is not true for non-polygons like a line or circle.

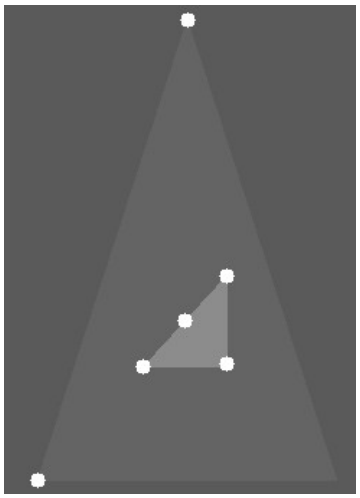
This idea was inspired by https://en.wikipedia.org/wiki/Ramer–Douglas–Peucker_algorithm

Explanation #4

To execute E.3, the algorithm checks all the pixels belonging to the shape and then finds the one that is farthest from the line. It is made slightly efficient by restricting the search in the y dimension. In image below, the algorithm searches for the triangle (indicated by red arrow) within the yellow lines. The beginning is identified during the segmentation step. The final y is identified as the last full line in which no pixels belonging to the triangle were found.

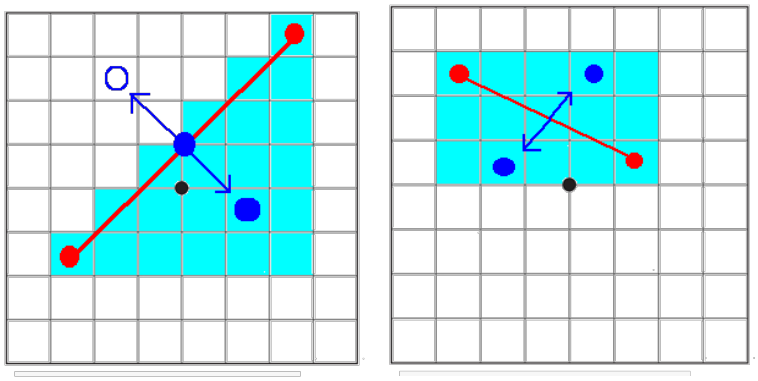


I tried to make the algorithm more efficient by following a path of increasing distance from the line, to limit the search space. But this was sensitive to where the path was initialized. An example of this error is given below. Notice that the inside triangle's vertices are identified correctly. But the outside triangle's third vertex path is blocked by the inside triangle, hence it is found against the inside triangle.



Explanation #5

To determine if the connection between two vertices is an edge, the pixel to the left and to the right of the midpoint is examined. One pixel – and only one pixel – should be “filled” (same value is the segmentation). This will always be true for all vertices of the triangle. But for quadrilaterals, both pixels will be filled, while for line segments neither pixel will be filled.



Below are what the results look like on one image. Black circles are vertices for each shape. Hollow white circles are those pixels that are examined to classify the edge.