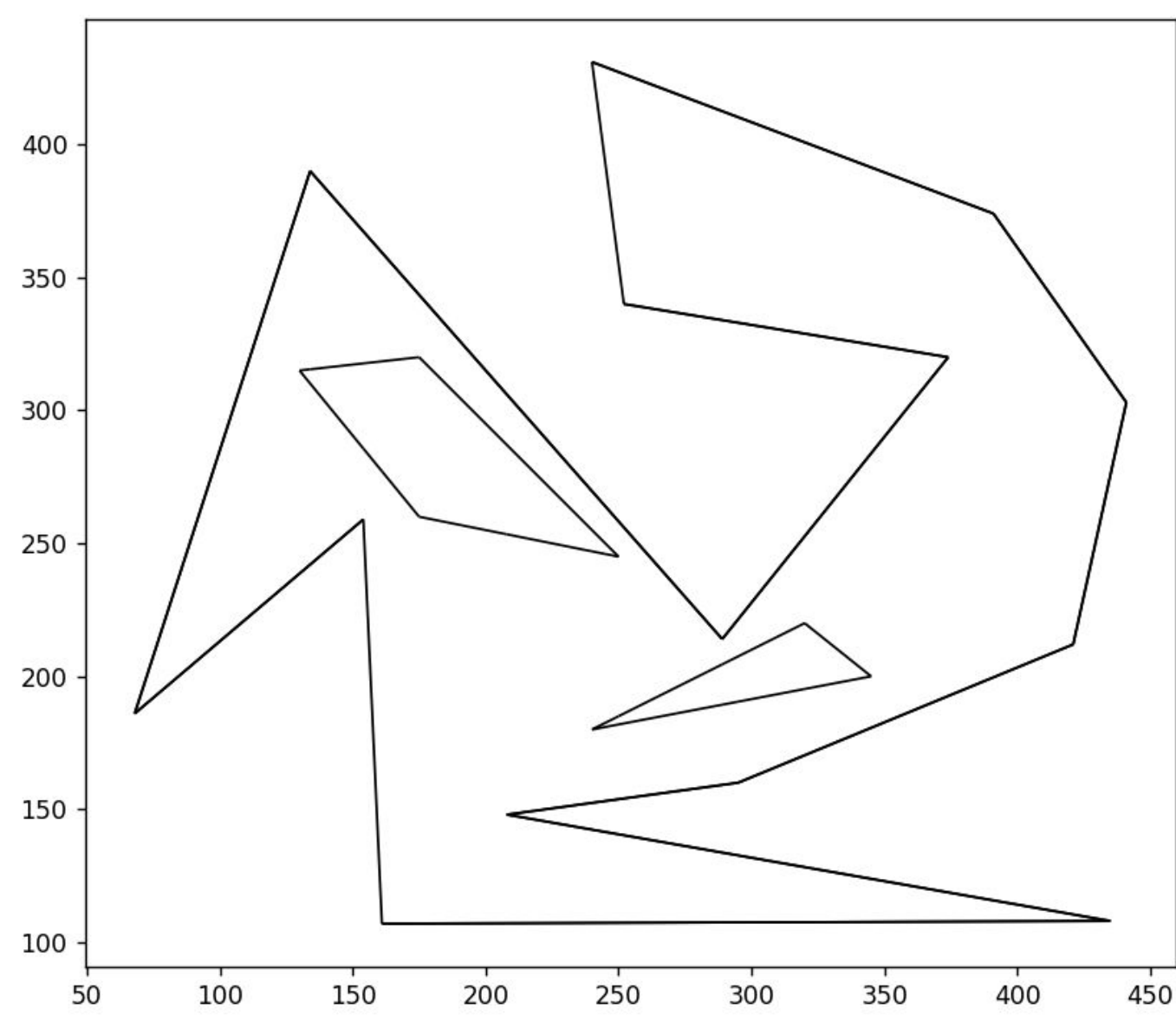


The WatchMan Problem

A watchman route is a path in a given area that allows a "watchman" to see every point in that area from at least one location on the route. The “problem” involves finding the shortest possible watchman route in the area with a certain amount of “routes” in the case of having multiple watchmen.

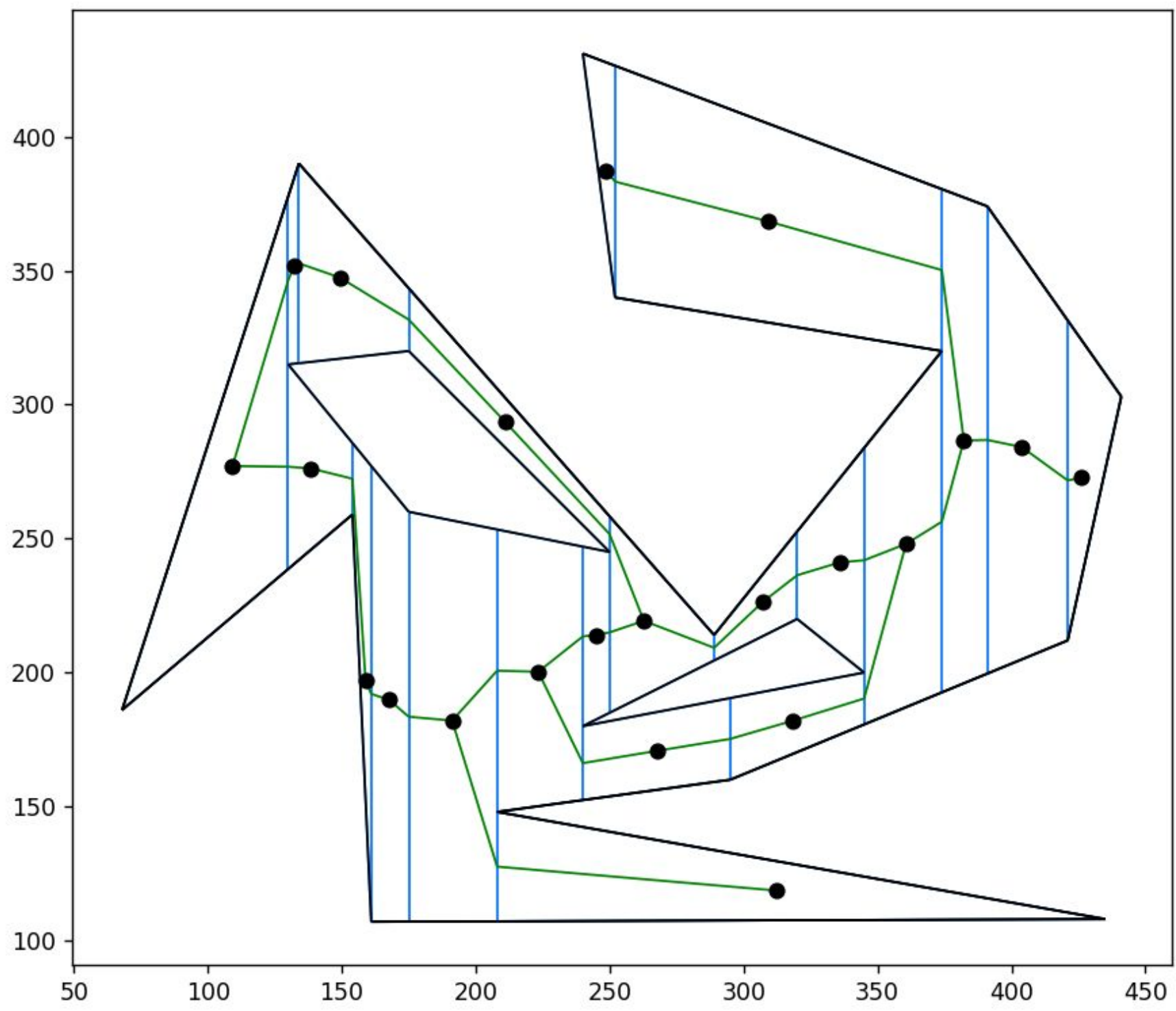
For this project, we will be using real life examples of lakes with obstacles and holes in the edges in them to compare our program with custom made polygon examples created by our team,also with holes and obstacles, that showcases both theoretical and real life uses for the watchman route.



Graph Creation and Tree Partitioning

Once the polygon is partitioned into trapezoids, all that’s left is making the graph. Connecting all the trapezoids together and representing them as a graph (where nodes are trapezoids, and edges are the sides they share) will be done by our python library.

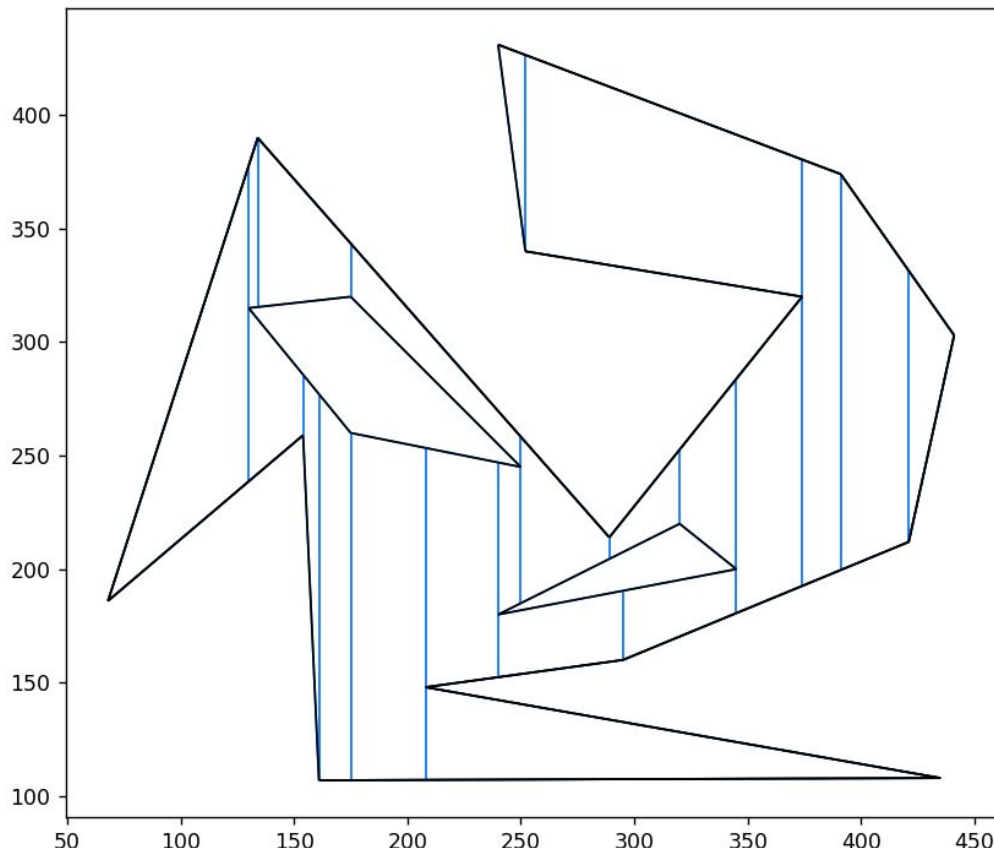
As is seen on the right, there is a point (representing a node) in each trapezoid, and there is a line connecting them (representing an edge). The edges are not always straight, as they need to traverse the midpoint of the edges that the trapezoids share.



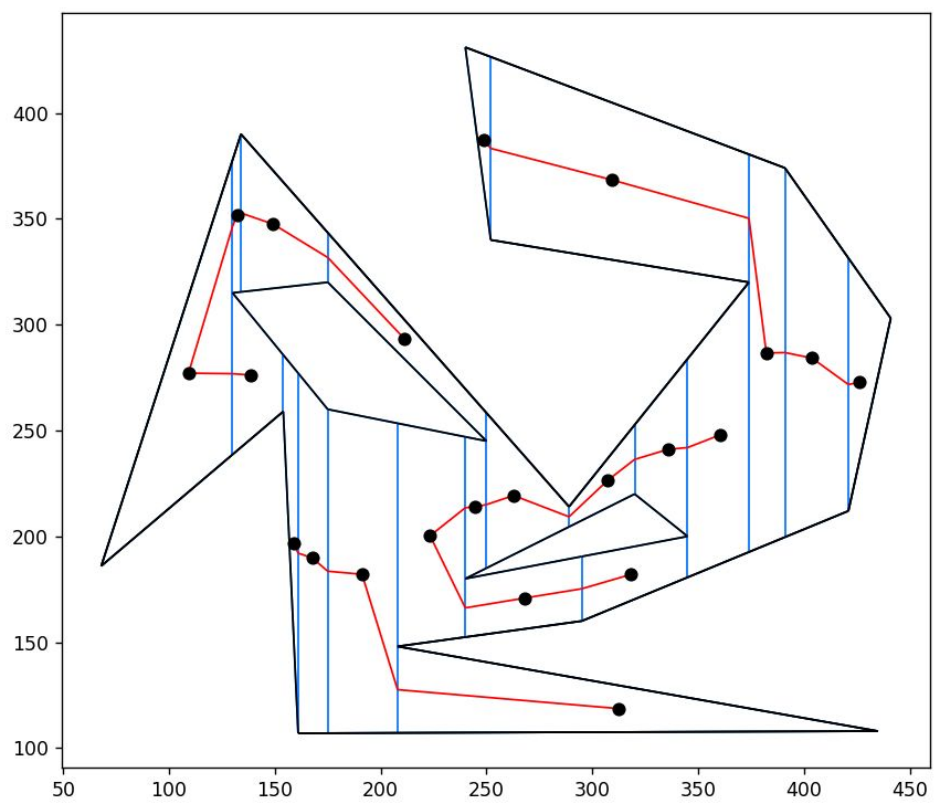
CGAL - Trapazoidation

CGAL is the Computational Geometry Algorithms Library, a C++ library that provides reliable geometric algorithms. After processing our input file, we utilized an algorithm to partition the shape into pseudo-trapezoids. The benefit of this is that these pseudo-trapezoids are a type of convex polygon, which is a subset of polygons with the property that (in our case) a watchman inside of a convex polygon can view the entire boundary of that complex polygon from any point.

This is the core of our algorithm, as once the area is divided into convex polygons, all we need to do is visit each new pseudo-trapezoid, and we would satisfy our objective.



Now that a graph exists, the final step is to obtain k subtrees for k boats to traverse. Firstly, we run Kruskal’s algorithm to obtain a minimum spanning tree (if there was k=1 boat, this is the answer, and the route length is twice the span of the tree). After that, we run a search algorithm to minimize the maximal subtree. After we reach the optimal partitioning for K subtrees, the solution is visualized with matplotlib.



K = 4 subtrees

On the right is the result of running the algorithm on Lake Urmia, Iran. With k=1

