Art Gallery Problem

Fangshuo Li

UNR CS302 FALL

## Intro

This project required two main parts: triangulation and tri-coloring. Explanations of code will go under these two sections.

As general information, I used the std::vectors for my data structure. I had two main vectors, one for the finished triangulated data, and one for storing the polygon as an array of points. The two structs, Point and Triangle fell under these vectors.

The execution of the project requires

make

to be ran first, which creates the executable main file. Then

./main

will run the program, which first prompts the user to enter how many points are on the polygon. Once entered, the program prompts the user to enter the points of the polygon one by one, in a clockwise fashion. For testing purposes, I used:

7 5

6 4

7 3

6 0

5 1

3 0

1 1

This yielded the results:

Text

Description automatically generated

The graph of the polygon is shown below (Note: the colors are arbitrary, but they do not line up with the key that is given in the code files) Chart, line chart

Description automatically generated

## Triangulation

In this project, I used the ear clipping method, mainly because of its ease to implement compared to other types. In this method, the main objective is to loop around the polygon and find triangles that are

1. Inside the polygon
2. Do not cross any other line segments

For this project, the ideas of concavity were heavily used in the form of clockwise and counterclockwise shapes. For example, the method to test if the triangle was inside the polygon was to find if the triangle was oriented clockwise or counterclockwise. If it was counterclockwise, it couldn’t be in the polygon since the polygon itself is clockwise.

To check if the triangle would intersect another line segment, it was tested if another vertex would be inside the triangle. This was again tested using clockwise or counterclockwise notation. Forming 3 sub-triangles with the testing point, it can be tested whether or not the point is in the triangle depending on if any of the sub-triangles were counterclockwise. If any sub-tree was counterclockwise, it meant that the point could not be inside the triangle.

Once a suitable triangle was found, the middle value, aka polygon[i+1] was removed from the vector and metaphorically “clipped”. Repeating this process and storing the valid triangles lead to the triangulation of the polygon.

## Tri-Coloring

This project used a very simple albeit slightly inefficient method to color the triangles. Because the triangles are stored in an array, it meant that incrementing through the triangles was made easier. Every point on the first triangle was colored. After that, the program loops through all triangles until it finds a triangle with 2 points colored. The 3rd tip becomes apparent, and the search for the next 2-point colored triangle begins. After an entire trip without coloring, the amount of each color is summed, and the lowest value represents the worst-case number of guards to guard the polygon.