Graphics Homework Assignment 1

Erin McGowan

Fall 2023

1 General Information

Operating system: macOS Big Sur version 11.5.2
Compiler: clang, specifically "C/C++: clang++ build and debug active file" in VSCode (as described here: https://code.visualstudio.com/docs/cpp/config-clang-mac)

2 Convex Hull

Task 1: Complete the function to compute if three points A, B, C form a salient angle (used to determine if the rope makes a right turn or not).

Given three points a, b, and c, we find the angle between them (with B as the middle point) using the following equation:

$$\theta = (y_b - y_a) * (x_c - x_b) - (x_b - x_a) * (y_c - y_b),$$

where x_a is the x-coordinate of point a, etc.
If the angle is greater than zero, these three angles form a right turn (this is because we have oriented "12 o'clock" in our conceptualization of counterclockwise at "3 o'clock" to align with the x-axis [see Task 2]).

Task 2: Complete the comparison structure to sort points in counter-clockwise order.

To sort the points in counterclockwise order with respect to the leftmost point with the lowest y coordinate, $p0$, we must first orient our "clock" so that "12 o'clock" in our conceptualization of counterclockwise at "3 o'clock" to align with the x-axis. To do so, we rotate each point as follows:

$$x = y$$
$$y = -x.$$

Next, we compute the angle θ between the vector extending from $p0$ to a given point in the point cloud and the vector extending from $p0$ along the x-axis.

θ= -1 * atan2(rotatedX1 - rotatedX0, -1 * (rotatedY1 - rotatedY0)),

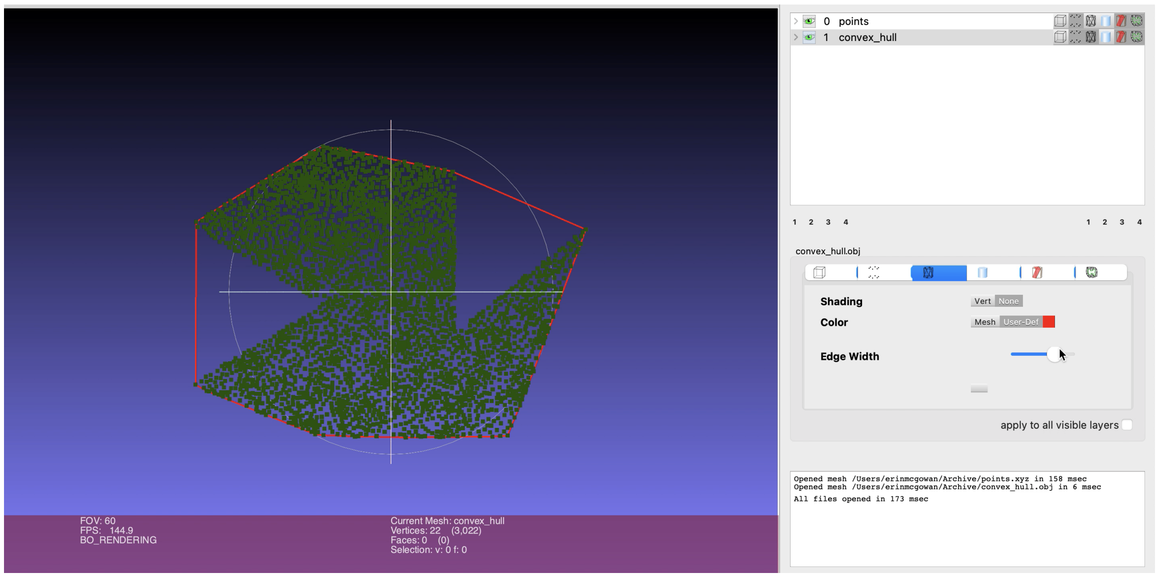
where rotatedX1 and rotatedY1 are the rotated coordinates of a given point in the cloud, and rotatedX0 and rotatedY0 are the rotated coordinates of $p0$.
We sort the points in increasing order by θ, which sorts the points in counterclockwise order.

Task 3: Implement Graham's scan algorithm using the information provided above.

To implement Graham's scan algorithm, we first identify the point in the point cloud with the lowest y-coordinate, storing each point with this minimum y value in a vector. If the length of this vector is more than one after iterating through each of the 3000 points, we choose the one with the lowest x-coordinate as our bottommost/leftmost point, $p0$. We then create a Compare struct (named *order*) and assign the value of *order.p*0 to this selected point. Next, we use the *std::sort* function to sort the points in *order* in counterclockwise order with respect to $p0$ using the method outlined in Task 2. Finally, we iterate through the sorted points, storing the first three into a polygon object called *hull* that we will be treating as a stack. For each of the remaining points, we check if the current point, the previous point, and the point immediately prior to the previous point form a right angle using the method outlined in Task 1. If they do, we remove the previous point (the central of the three points) and repeat this process until the three points do not form a right angle. After repeating this process for each of the sorted points, the object *hull* contains the convex hull outlining the given point cloud.

Task 4: Complete the function to read .xyz from a file.

We first read in each line of the points.xyz file using the $c++$ *getline* function. We skip the first line (containing the number of points). For each subsequent line, we iterate through the characters until we reach a space. This denotes the end of the x coordinate for that point, which we save to a Point object, casting it as a double. We then continue to iterate through the characters of this line until we reach another space. This denotes the end of the y coordinate for that point; we save the characters between the previous space and this new space to a Point object, casting it as a double.



```
FOV: 60
FPS:   144.9
BO_RENDERING
```

```
Current Mesh: convex_hull
Vertices: 22    (3,022)
Faces: 0    (0)
Selection: v: 0 f: 0
```

```
Opened mesh /Users/erinmcgowan/Archive/points.xyz in 158 msec
Opened mesh /Users/erinmcgowan/Archive/convex_hull.obj in 6 msec
All files opened in 173 msec
```

3 Point in Polygon

Task 1: Write the functions to load an OBJ polygon, and to save an XYZ point cloud.

We load points.xyz using the same function described in Task 4 of the Convex Hull problem. We take a similar approach to loading the polygon.obj file, however we do not skip the first line of the file, and we only iterate through lines beginning with "v". Also, we skip the first two characters, "v ", of each line. We save the resulting vector of points to a .xyz file, with the length of the point vector saved to the first line of the output file and each point saved to a separate line in "x_coordinate y_coordinate" format.

Task 2: Write the function to determine whether two segments intersect, and compute the location of the intersection.

In order to determine whether two line segments intersect, we first take the determinant of the four endpoints (one at each end of each line segment) by treating them as a matrix of vectors and rearranging the entries as follows:

$$determinant = (y_b - y_a) * (x_c - x_d) - (y_d - y_c) * (x_a - x_b),$$

where one line segment is between points a and b and the other is between points c and d.
If the determinant is equal to zero, the two line segments are parallel and do not intersect. If the determinant is not zero, we solve for the intersection point of the two lines (one for each line segment, each of which goes to infinity at either end), rearranging the equations as follows:

$$x = \frac{((x_c - x_d) * ((y_b - y_a) * (x_a) + (x_a - x_b) * (y_a)) - (x_a - x_b) * ((y_d - y_c) * (x_c) + (x_c - x_d) * (y_c)))}{determinant}$$
$$y = \frac{((y_b - y_a) * ((y_d - y_c) * (x_c) + (x_c - x_d) * (y_c)) - (y_d - y_c) * ((y_b - y_a) * (x_a) + (x_a - x_b) * (y_a)))}{determinant},$$

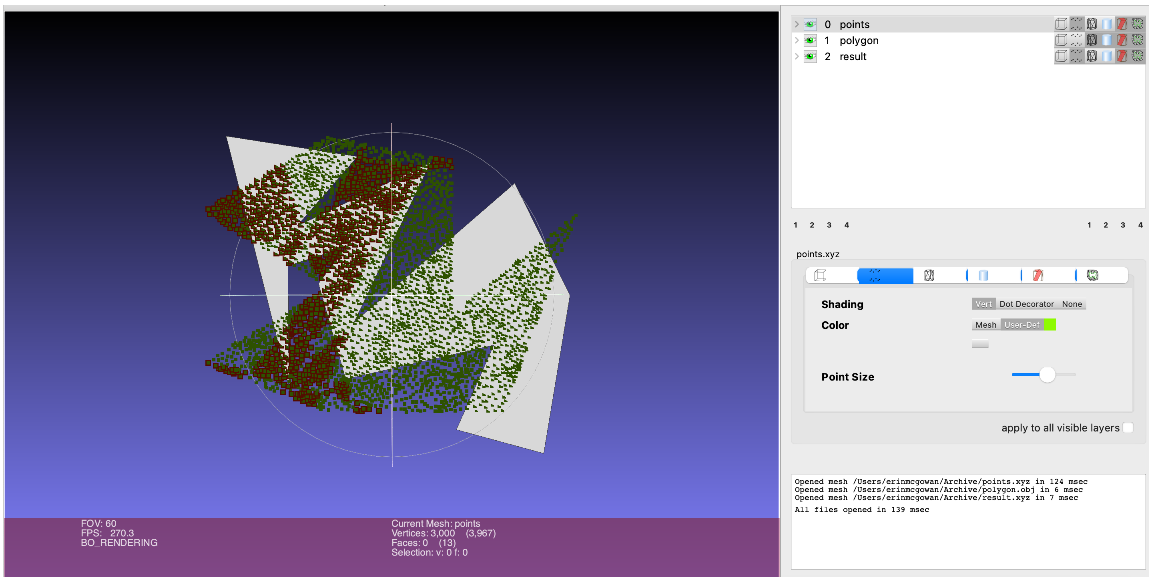
where x_a is the x-coordinate of point a, etc.
Finally, we determine if this point of intersection lies on the given line segments (i.e. between our four points of interest). To do so, we find the maximum and minimum x and y-coordinate values amongst the four endpoints, and check if the x coordinate of the point of intersection lies between the minimum and maximum x values and the y coordinate of the point of intersection lies between the minimum and maximum y values. If it does, the two line segments intersect and we return true. If it does not, the line segments themselves do not intersect (even though they are not parallel) and we return false.

Task 3: Compute the bounding box of the input polygon, then find a point that is guaranteed to be outside the input polygon.

To compute the bounding box of the input polygon, we find the maximum x coordinate, minimum x coordinate, maximum y coordinate, and minimum y coordinate within the set of polygon vertices. We then use (minimum x, maximum y), (maximum x, maximum y), (maximum x, minimum y), and (minimum x, minimum y) as the four corners of the polygon's bounding box. We then set ($2*$ maximum x, $2*$ maximum y) as our point guaranteed to be outside the input polygon.

Task 4: Write the Point In Polygon function.

After loading the point cloud and polygon objects, we iterate through each of the 3000 points in the point cloud. For each point, we check whether the line segment between said point and the point guaranteed to be outside the polygon (computed in Task 3) intersections each of the 14 edges of the polygon (each represented by the pair of points at either end) using the method outlined in Task 2. If the line segment intersects with an even number of edges, the point in question lies outside of the polygon. If the line intersects with an odd number of edges, the point in question lies inside the polygon.



```
FOV: 60
FPS:   270.3
BO_RENDERING
```

```
Current Mesh: points
Vertices: 3,000    (3,967)
Faces: 0    (13)
Selection: v: 0 f: 0
```

```
Opened mesh /Users/erinmcgowan/Archive/points.xyz in 124 msec
Opened mesh /Users/erinmcgowan/Archive/polygon.obj in 6 msec
Opened mesh /Users/erinmcgowan/Archive/result.xyz in 7 msec
All files opened in 139 msec
```